

Docket No.: 67161-063

**PATENT**

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

In re Application of

Atsuo YAMAGUCHI

Serial No.:

Group Art Unit:

Filed: July 09, 2003

Examiner:

For: RESIDUE CALCULATING UNIT IMMUNE TO POWER ANALYSIS

**CLAIM OF PRIORITY AND  
TRANSMITTAL OF CERTIFIED PRIORITY DOCUMENT**

Mail Stop Patent Application  
Commissioner for Patents  
P.O. Box 1450  
Alexandria, VA 22313-1450

Sir:

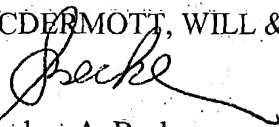
In accordance with the provisions of 35 U.S.C. 119, Applicant hereby claims the priority of:

**Japanese Patent Application No. 2002-286182, filed September 30, 2002,**

cited in the Declaration of the present application. A certified copy is submitted herewith.

Respectfully submitted,

MCDERMOTT, WILL & EMERY

  
Stephen A. Becker  
Registration No. 26,527.

600 13<sup>th</sup> Street, N.W.  
Washington, DC 20005-3096  
(202) 756-8000 SAB:km  
Facsimile: (202) 756-8087  
**Date: July 9, 2003**

日本国特許庁

JAPAN PATENT OFFICE

67161-063  
Atsuo Yamaguchi  
July 9, 2003

McDermott, Will & Emery

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office

出願年月日

Date of Application:

2002年 9月30日

出願番号

Application Number:

特願2002-286182

[ST.10/C]:

[JP2002-286182]

出願人

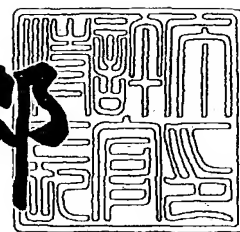
Applicant(s):

三菱電機株式会社

2002年10月25日

特許庁長官  
Commissioner,  
Japan Patent Office

太田信一郎



出証番号 出証特2002-3083878

【書類名】 特許願

【整理番号】 539518JP01

【提出日】 平成14年 9月30日

【あて先】 特許庁長官殿

【国際特許分類】 G09C 1/00  
G06F 7/72  
H04L 9/30

【発明者】

【住所又は居所】 東京都千代田区丸の内二丁目2番3号 三菱電機株式会社  
社内

【氏名】 山口 敦男

【特許出願人】

【識別番号】 000006013

【氏名又は名称】 三菱電機株式会社

【代理人】

【識別番号】 100064746

【弁理士】

【氏名又は名称】 深見 久郎

【選任した代理人】

【識別番号】 100085132

【弁理士】

【氏名又は名称】 森田 俊雄

【選任した代理人】

【識別番号】 100083703

【弁理士】

【氏名又は名称】 仲村 義平

【選任した代理人】

【識別番号】 100096781

【弁理士】

【氏名又は名称】 堀井 豊

【選任した代理人】

【識別番号】 100098316

【弁理士】

【氏名又は名称】 野田 久登

【選任した代理人】

【識別番号】 100109162

【弁理士】

【氏名又は名称】 酒井 將行

【手数料の表示】

【予納台帳番号】 008693

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 ベキ乗剰余演算器

【特許請求の範囲】

【請求項 1】 第 1 の種類のデータを保持する第 1 のレジスタ群と、  
前記第 1 のレジスタ群に保持されたデータと並行して参照可能な種類のデータを保持する第 2 のレジスタ群と、  
前記第 1 のレジスタ群に接続された第 1 の内部バスと、  
前記第 2 のレジスタ群に接続された第 2 の内部バスと、  
前記第 1 および第 2 の内部バスに接続され、前記第 1 および第 2 のレジスタ群に保持された値を並行して参照し、モンゴメリ乗算剰余演算を実行するためのモンゴメリ乗算剰余演算実行手段と、  
前記第 1 および第 2 の内部バス、ならびに前記モンゴメリ乗算剰余演算実行手段に接続され、前記第 1 および第 2 のレジスタ群に保持された値を並行して参照し、前記モンゴメリ乗算剰余演算実行手段との間でデータのやり取りを行ない、ベキ乗剰余演算を実行するためのベキ乗剰余演算実行手段と、  
前記モンゴメリ乗算剰余演算および前記ベキ乗剰余演算の各計算結果を得るためには省略可能な中間演算処理を擬似的に実行するための擬似演算実行手段とを備える、ベキ乗剰余演算器。

【請求項 2】 前記擬似演算実行手段は、  
前記第 1 の内部バスに接続され、バイナリ法に従う前記ベキ乗剰余演算を実行する際に、破棄されるべき中間演算結果を一旦格納するためのダミーレジスタを含む、請求項 1 記載のベキ乗剰余演算器。

【請求項 3】 前記第 1 のレジスタ群は、  
バイナリ法に従う前記ベキ乗剰余演算を実行する際に、前記ベキ乗剰余演算の中間結果を格納するための中間結果格納レジスタを含み、  
前記擬似演算実行手段は、  
前記中間結果格納レジスタの初期値を単位元とする代わりに、ベキ乗を表す 2 進整数のビット値のうち、最上位ビットから最初に 1 になった時の前記中間結果を、前記ベキ乗のベキ乗される数の値に置換える置換手段を含む、請求項 1 記載

のべき乗剰余演算器。

【請求項4】 前記置換手段は、前記モンゴメリ乗算剰余の補正演算において、前記中間結果格納レジスタに格納されるべき前記中間結果の演算において、オペランドを0と前記べき乗される数に変更して前記中間結果格納レジスタに格納することにより、前記中間結果を前記べき乗のべき乗される数の値に置換える、請求項3記載のべき乗剰余演算器。

【請求項5】 前記第2のレジスタ群は、  
前記モンゴメリ乗算剰余演算において、繰り返し累積加算をしていく途中の値を格納する累積加算レジスタを含み、  
前記擬似演算実行手段は、  
前記累積加算レジスタの初期値を零元とする代わりに、乗数の各ビット値のうち最下位ビットから最初に1になった時点で、前記累積加算レジスタの値に代えて値0を前記累積加算レジスタの値として読み出す読出し手段を含む、請求項1記載のべき乗剰余演算器。

【請求項6】 前記第2のレジスタ群は、  
前記モンゴメリ乗算剰余演算において、繰り返し累積加算をしていく途中の値を格納する累積加算レジスタを含み、  
前記モンゴメリ乗算剰余演算実行手段は、  
前記モンゴメリ乗算剰余演算と前記モンゴメリ乗算剰余演算における補正演算の双方に共用して使用される累積加算器を含み、  
前記擬似演算実行手段は、  
前記補正演算の要否とは独立して、前記累積加算レジスタへの書き込み動作を行うためのレジスタ入出力手段を含む、請求項1記載のべき乗剰余演算器。

【請求項7】 前記レジスタ入出力手段は、  
前記補正演算の結果に対して右シフト処理を行って前記累積加算レジスタに書込むための右シフト手段と、  
前記右シフト処理において、前記補正演算の結果の最下位ビットを保持するための一時保持レジスタと、  
前記累積加算レジスタからの読出しにおいて、読み出された値を左シフトし、

かつ、左シフト結果に前記一時保持レジスタに保持された値を最下位ビットとして付加するための左シフト手段とを含む、請求項6記載のべき乗剰余演算器。

【請求項8】 前記第2のレジスタ群は、

前記モンゴメリ乗算剰余演算において、繰り返し累積加算をしていく途中の値を格納する累積加算レジスタと、

ダミーレジスタとを含み、

前記モンゴメリ乗算剰余演算実行手段は、

前記モンゴメリ乗算剰余演算と前記モンゴメリ乗算剰余演算における補正演算の双方に共用して使用される累積加算器を含み、

前記擬似演算実行手段は、

前記補正演算が必要な場合は、前記累積加算レジスタへの書き込み動作を行い、前記補正演算が不要な場合は、前記ダミーレジスタへの書き込み動作を行なうためのレジスタ入出力手段を含む、請求項1記載のべき乗剰余演算器。

【請求項9】 破棄されるべき中間演算結果を計算する時の演算の入力を与え、前記破棄されるべき中間演算結果を一旦格納するためのダミーレジスタをさらに備える、請求項1記載のべき乗剰余演算器。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、情報通信ネットワーク、交通、金融、医療、流通等の分野において使用される情報の暗号化技術を実現するための乗剰余演算器に関し、特に、モンゴメリ (Montgomery) のアルゴリズムを用いてべき乗剰余演算を行なうべき乗剰余演算器の構成に関する。

【0002】

【従来の技術】

情報通信技術の発展に伴い、情報ネットワーク上のセキュリティの確保（データの盗用や破壊を防止すること）が重要視されるようになってきている。そのため、情報の暗号化技術および復号化技術が採用されることが多く、その適用分野は単なる情報通信分野に留まらず、交通、金融、医療、流通等の身近な分野にま

で広がりつつある。この種の暗号化技術および復号化技術には、高度なセキュリティを単純な原理によって実現できることが要求される。

#### 【0003】

まず、この種の技術の理解を容易にするため、情報の暗号化・復号化についての概略を説明する。暗号の世界においては、“非対称暗号アルゴリズム”が質的に優れている。非対称暗号アルゴリズムとは、暗号化鍵と復号化鍵とが異なり、そのいずれか一方から他方が“容易に計算できない”暗号アルゴリズムをいう。この非対称暗号アルゴリズムの代表的なものに、べき乗剰余演算（ある数  $X$  を何回も乗算して  $N$  で割った余りをとる計算）を用いるタイプの RSA (Rivest-Shamir-Adleman scheme) 暗号がある。

#### 【0004】

RSA暗号を生成するには、次式(1)のべき乗剰余演算の形式が基本として使用される。式(1)は、 $XY$ を $N$ で割ったときの余りを求めることを意味する。また、式(1)において、 $X$ は暗号化(復号化)の対象となる平文、 $Y$ および $N$ は暗号化(復号化)のための鍵(キー)である。

#### 【0005】

$$X^Y \bmod N \quad \dots (1)$$

このべき乗剰余演算を用いることにより、情報の暗号化および復号化が容易に実行され、かつ $X$ 、 $Y$ 、 $N$ のオペランドビット長を長くすることで、各鍵の解読を困難にすることができる。

#### 【0006】

しかし、オペランドビット長を長くすると、べき乗剰余演算に長時間を要することになる。そこで、オペランドビット長が長いべき乗剰余演算をいかに短時間に終了させるかが1つのポイントとなる。

#### 【0007】

次に、RSA暗号を例に取り、べき乗剰余演算を使用した暗号化処理および復号化処理について説明する。

#### 【0008】

[RSA暗号の暗号化および復号化について]



RSA暗号の暗号化には、次式(2)が用いられる。

【0009】

$$C = M^e \bmod N \quad \dots (2)$$

復号化には、次式(3)が用いられる。

【0010】

$$M = C^d \bmod N \quad \dots (3)$$

ここで、Mは暗号化の対象となる平文、Cは暗号化された平文すなわち暗号文である。また、式(2)におけるeおよびNは暗号化鍵、式(3)におけるdおよびNは復号化鍵である。また、以下の式(4)および式(5)の関係が予め与えられている。

【0011】

$$N = p \cdot q \quad \dots (4)$$

$$1 \equiv e \cdot d \bmod \{ \text{LCM}(p-1, q-1) \} \quad \dots (5)$$

ここで、「 $\equiv$ 」は、左辺と右辺とが相似であることを意味し、「LCM」は、最小公倍数を意味する。またpとqとは互いに素な整数である。なお、eおよびNは公開鍵であり、d、pおよびqは秘密鍵である。

【0012】

式(4)および式(5)は、ともに暗号アルゴリズムにおけるべき乗剰余演算の数値の条件を定義している。式(4)は、Nは互いに素な大きな素数pおよびqの積であることを示している。pおよびqはともに奇数なので、当然Nは奇数でなければならない。式(5)は、式(4)で示したpおよびqからそれぞれ1を減じた値同士の最小公倍数で、eおよびdの積 $e \cdot d$ を当該最小公倍数で割ったときの余りが1になることを示している。

【0013】

式(4)および式(5)の条件に基づき、平文Mは式(2)を用いて暗号化され、また暗号化された平文M(暗号文C)は式(3)を用いて復号化される。

【0014】

[べき乗剰余演算の演算方法について]

次に暗号化・復号化で使用する、べき乗剰余演算の演算方法を説明する。A

$=M^e \bmod N$  のべき乗剰余演算は、整数  $e$  の 2 進数展開を  $e = e^{k-1} \dots e^1 e^0$  として、以下のフロー 1 に示す反復平方積法を用いることにより実行される。

【0015】

(フロー 1)

begin

$A = 1$

for  $i = k - 1$  to 0

begin

$A = A^2 \bmod N \quad \dots (6)$

If  $e^i = 1$  then  $A = A \cdot M \bmod N \quad \dots (7)$

end

end

A に格納された値が求めたいべき乗剰余演算の解になる。

【0016】

以上のように演算の基本は、式 (6) および式 (7) に示すように乗算と除算 (mod 算) である。乗算は、初期値を 1 とする A の値に対して  $A \times A$  または  $A \times M$  を行なう部分である。除算は、各々の乗算で得られた値に対して mod N (N で割ったときの余りを求める演算) を行なう部分である。この“乗算と除算” ( $A \times A \bmod N$ 、 $A \times M \bmod N$ ) を 1 対の演算として、“e” のビット値に従って繰返し演算が行なわれる。すなわち、“e” の最上位ビットから最下位ビットまでの各ビットの内容によって“乗算と除算”が行なわれる。

【0017】

べき乗剰余演算は、基本となる剰余演算 (mod 算) を繰返し行なうことで解が得られることを示したが、この繰返し回数自体は、たかだか数百～数千回であるので、ソフトウェアによる処理でも十分に対応することも可能である。

【0018】

しかし、この剰余演算自体すなわち除算をハードウェアによって実行するためには、大規模な演算回路と複雑な処理手順とが必要とされる。このため改善が望まれていた。通常、 $e$ 、 $d$ 、 $M$ 、 $N$  などは 1024 ビット程度の大きな整数が用

いられているので、高速指数計算法を使用しても1回のRSA演算で平均1500回程程度の多重精度乗算と剰余算とを行なわなければならない。特に剰余算は、近似法、剰余テーブル方式、モンゴメリのアルゴリズム等、多くの高速化手法が提案されている。

#### 【0019】

このような、RSA暗号に代表される公開鍵暗号の多くで利用される、べき乗剰余演算を高速に処理するためには、1回当りの剰余算の高速化が要求される。モンゴメリのアルゴリズムは、剰余算を高速処理するアルゴリズムである。特に、乗算剰余演算においては、除算をビットシフトなどで簡略化できるため、公開鍵暗号（RSA暗号等）で用いられるべき乗剰余演算を高速処理することができるという特徴がある。

#### 【0020】

一方、中国人の剰余定理により、合成数を法とする演算は合成数を構成する互いに素な因数を法とする演算から計算できる。これを1024ビット長RSA暗号処理に適用すると、実際に必要なハードウェアとしては、1024ビット長の法Nによるべき乗剰余演算器ではなく、512ビット長の整数（ここではpおよびqに相当する）を法とする演算回路のみでよい。このためハードウェアの小型化に繋がる。

#### 【0021】

べき乗剰余演算は、基本となる剰余演算(mod算)を実行する手順が非常に複雑であるため、演算回路が大規模化してしまうことを上述した。そこで、モンゴメリは、剰余演算(mod算)を先のような一般的な方法で行わずに、“乗算”と簡単なビット列処理とで行なうことによって解を得る仕組みを提案している。以下にモンゴメリが提案している手法について簡単に説明する。

#### 【0022】

##### [モンゴメリのアルゴリズム]

剰余演算の高速化を実現する一手法であるモンゴメリのアルゴリズムについて説明する。

#### 【0023】

モンゴメリのアルゴリズムは、剰余の法 $N$  ( $N > 1$ ) と、剰余の法 $N$ と互いに素である定数 $R$  ( $R > N$ ) とを用いると、被剰余数を $T$ とした場合、 $TR^{-1} \bmod N$ の計算が定数 $R$ による除算のみで行なえる性質を利用している。これにより、 $N$ による除算を用いることなく剰余計算を行なうことができる。

【0024】

ここで、 $N$ 、 $R$ 、 $R^{-1}$ および $T$ は整数である。被剰余数 $T$ は $0 \leq T < R \cdot N$ を満たす数である。 $R^{-1}$ は剰余の法 $N$ の上での定数 $R$ の逆数である。またここさらに、 $R \cdot R^{-1} - N \cdot N' = 1$  ( $0 \leq R^{-1} < N$ 、 $0 \leq N' < R$ ) の関係を満たす整数 $N'$  を考えることができる。さらに、この定数 $R$ に2のべき乗数を使用した場合、定数 $R$ による除算をシフト操作に置き換えることができる。このため、 $T \rightarrow TR^{-1} \bmod N$  (被剰余数を $T$ とした場合の $TR^{-1} \bmod N$ ) の計算の高速処理が可能となる。

【0025】

次にアルゴリズム1として、 $T \rightarrow TR^{-1} \bmod N$ のアルゴリズム $MR(T)$ を示す。ただし、アルゴリズム1において $(T + m \cdot N) / R$ は必ず割り切れることが証明されている。

【0026】

(アルゴリズム1)  $T \rightarrow TR^{-1} \bmod N$ のアルゴリズム $Y = MR(T)$ は次のように表わされる。

【0027】

$$M = (T \bmod R) \cdot N' \bmod R \quad \dots (8)$$

$$Y = (T + m \cdot N) / R \quad \dots (9)$$

if  $Y \geq N$  then  $Y = Y - N$

$Y < N$  then return  $Y$

1回の $MR$ では、剰余 $T \bmod N$ ではなく $TR^{-1} \bmod N$ が求められるだけである。よって、剰余 $T \bmod N$ を求めるためには、次に示すように $MR(T)$ と予め求めておいた $R^2 \bmod N$ との積で、再び $MR$ 演算を行なえばよい。

【0028】

$$MR(MR(T) \cdot (R^2 \bmod N))$$

$$\begin{aligned}
 &= (TR^{-1} \bmod N) \cdot (R^2 \bmod N) \cdot R^{-1} \bmod N \\
 &= TR^{-1} \cdot R^2 \cdot R^{-1} \bmod N \\
 &= T \bmod N
 \end{aligned}$$

このようにして剰余  $T \bmod N$  を求めることができる。

#### 【0029】

以上のことにより、モンゴメリ法による乗算剰余演算を使用して、これをべき乗剰余演算の反復平方積法（繰返し2乗法）で実現するアルゴリズムを下記に示す。鍵  $e$  の上位ビットから検索し、鍵のビットの値が1の場合には、 $MR(X \cdot Y)$  のモンゴメリ乗算剰余演算を行なう。

#### 【0030】

```

Y = R r    (R r = R2 mod N (R = 2k+2))
X = M
X = MR (X · Y)      ... (10)
Y = MR (1 · Y)      ... (11)
for j = k to 1
    if ej == 1 then Y = MR (X · Y)      ... (12)
    if j > 1 then Y = MR (Y · Y)      ... (13)
end for
Y = MR (1 · Y)      ... (14)
Y = Y mod N      ... (15)

```

ここで、 $MR(X \cdot Y)$  と  $MR(Y \cdot X)$  とは等しく、 $e_j$  は鍵  $e$  の  $j$  ビット目を表わす。512ビット長の整数の場合  $k = 512$  となり、512ビットのべき乗剰余演算は514ビットのモンゴメリ乗算剰余演算と512ビット剰余演算とにより実現できる。

#### 【0031】

また、ハードウェアとして実装するのに最適な基数  $W$  の逐次計算でモンゴメリ乗算剰余演算結果  $P = MR(B \cdot A)$  を求めると、以下のようなになる。

#### 【0032】

$$W = 2^d$$

$$N0' = N' \bmod W$$

$$P = 0$$

for j = 0 to k

$$M = (P \bmod W) \cdot N0' \quad \dots (16)$$

$$P = (P + (A \bmod W) \cdot B \cdot W + M \cdot N) / W \quad \dots (17)$$

$$A = A / W \quad \dots (18)$$

End

dは自然数であり、ハードウェアに依存する数である。このようにして、モンゴメリ乗算剰余演算結果Pを求めることができる。d=1の基数2の逐次計算で514ビットモンゴメリ乗算剰余演算結果P=MR(B・A)を求めると、以下のようなになる。

【0033】

$$N0' = N' \bmod 2$$

$$P = 0$$

for j = 0 to 514

$$M = (P \bmod 2) \cdot N0' \quad \dots (19)$$

$$P = (P + (A \bmod 2) \cdot B \cdot 2 + M \cdot N) / 2 \quad \dots (20)$$

$$A = A / 2 \quad \dots (21)$$

end

以上のように、べき乗剰余演算を実現するために、ハードウェアで512ビット長のべき乗剰余演算にモンゴメリ法を使用し、ソフトウェアで中国人の剰余定理を利用した処理を使用することが従来採用されている。ハードウェアへの実装方式は複数通りあり、実際に様々な方式が採用されていると思われる。

【0034】

RSA暗号についての従来技術としては、RSA暗号の説明、およびモンゴメリ法の説明および補正装置の説明が、たとえば、特許文献1に開示されており、RSA暗号の説明およびモンゴメリ法の説明が、たとえば、特許文献2に開示さ

れている。

【0035】

さらに、RSA暗号の説明、楕円暗号の説明およびモンゴメリ法の説明が、たとえば、特許文献3に開示されている。

【0036】

【特許文献1】

特開平7-20778号公報明細書

【0037】

【特許文献2】

特開平11-212456号公報明細書

【0038】

【特許文献3】

特許第2937982号公報明細書

【0039】

【発明が解決しようとする課題】

ところで、以上のようなアルゴリズムを用いて暗号化を行うハードウェアに対しては、その暗号化の鍵を外部から探索するにあたり、いわゆる「電力解析」と呼ばれる手法が用いられる場合がある。

【0040】

たとえば、パソコン、携帯電話、ICカードなど暗号を搭載する機器において、暗号するための鍵を探索するために、LSIの消費電流を見る手法として、単純電力解析(Simple Power Analysis: SPA)、電力差分解析(Differential Power Analysis: DPA)等が知られている。

【0041】

ここで、上述したようなRSA暗号の暗号化処理を行うためのハードウェアは、電力解析に対する耐性を可能な限り向上させることが望ましい。

【0042】

このためには、ハードウェアの消費電流パターンに特徴が少なく、内部のアルゴリズムが推定できないことが望ましい。さらに、外部から制御できない秘密デ

ータに依存した電流パターンが存在しないことが望ましい。また、外部から制御可能な入力データと消費電流値の相関が少なく、かつ、外部から制御可能な入力データと消費電流値の相関の大小のパターンに特徴が少ないことが望ましい。

【0043】

上述した、べき乗剰余演算においては、まず第一に、“e”のビット値単位で処理が行われる構成であるため、そのビット値が“1”であるか“0”であるかによって消費電流が変わらないことが望ましい。

【0044】

本発明は上述の課題を解決するためになされたもので、その目的は、回路規模が小さく、高速処理が可能であって、かつ、電力解析などに対する安全性を高めた暗号処理回路等に適用可能なべき乗剰余演算器を提供することである。

【0045】

【課題を解決するための手段】

請求項1記載のべき乗剰余演算器は、第1の種類データを保持する第1のレジスタ群と、第1のレジスタ群に保持されたデータと並行して参照可能な種類のデータを保持する第2のレジスタ群と、第1のレジスタ群に接続された第1の内部バスと、第2のレジスタ群に接続された第2の内部バスと、第1および第2の内部バスに接続され、第1および第2のレジスタ群に保持された値を並行して参照し、モンゴメリ乗算剰余演算を実行するためのモンゴメリ乗算剰余演算実行手段と、第1および第2の内部バス、ならびにモンゴメリ乗算剰余演算実行手段に接続され、第1および第2のレジスタ群に保持された値を並行して参照し、モンゴメリ乗算剰余演算実行手段との間でデータのやり取りを行ない、べき乗剰余演算を実行するためのべき乗剰余演算実行手段と、モンゴメリ乗算剰余演算およびべき乗剰余演算の各計算結果を得るためには省略可能な中間演算処理を擬似的に実行するための擬似演算実行手段とを備える。

【0046】

請求項2記載のべき乗剰余演算器は、請求項1記載のべき乗剰余演算器の構成に加えて、擬似演算実行手段は、第1の内部バスに接続され、バイナリ法に従うべき乗剰余演算を実行する際に、破棄されるべき中間演算結果を一旦格納するた



めのダミーレジスタを含む。

【0047】

請求項3記載のべき乗剰余演算器は、請求項1記載のべき乗剰余演算器の構成に加えて、第1のレジスタ群は、バイナリ法に従うべき乗剰余演算を実行する際に、べき乗剰余演算の中間結果を格納するための中間結果格納レジスタを含み、擬似演算実行手段は、中間結果格納レジスタの初期値を単位元とする代わりに、べき乗を表す2進整数のビット値のうち、最上位ビットから最初に1になった時の中間結果を、べき乗のべき乗される数の値に置換える置換手段を含む。

【0048】

請求項4記載のべき乗剰余演算器は、請求項3記載のべき乗剰余演算器の構成に加えて、置換手段は、モンゴメリ乗算剰余の補正演算において、中間結果格納レジスタに格納されるべき中間結果の演算において、オペランドを0とべき乗される数に変更して中間結果格納レジスタに格納することにより、中間結果をべき乗のべき乗される数の値に置換える。

【0049】

請求項5記載のべき乗剰余演算器は、請求項1記載のべき乗剰余演算器の構成に加えて、第2のレジスタ群は、モンゴメリ乗算剰余演算において、繰り返し累積加算をしていく途中の値を格納する累積加算レジスタを含み、擬似演算実行手段は、累積加算レジスタの初期値を零元とする代わりに、乗数の各ビット値のうち最下位ビットから最初に1になった時点で、累積加算レジスタの値に代えて値0を累積加算レジスタの値として読み出す読出し手段を含む。

【0050】

請求項6記載のべき乗剰余演算器は、請求項1記載のべき乗剰余演算器の構成に加えて、第2のレジスタ群は、モンゴメリ乗算剰余演算において、繰り返し累積加算をしていく途中の値を格納する累積加算レジスタを含み、モンゴメリ乗算剰余演算実行手段は、モンゴメリ乗算剰余演算とモンゴメリ乗算剰余演算における補正演算の双方に共用して使用される累積加算器を含み、擬似演算実行手段は、補正演算の要否とは独立して、累積加算レジスタへの書き込み動作を行うためのレジスタ入出力手段を含む。

## 【0051】

請求項7記載のべき乗剰余演算器は、請求項6記載のべき乗剰余演算器の構成に加えて、レジスタ入出力手段は、補正演算の結果に対して右シフト処理を行って累積加算レジスタに書込むための右シフト手段と、右シフト処理において、補正演算の結果の最下位ビットを保持するための一時保持レジスタと、累積加算レジスタからの読出しにおいて、読み出された値を左シフトし、かつ、左シフト結果に一時保持レジスタに保持された値を最下位ビットとして付加するための左シフト手段とを含む。

## 【0052】

請求項8記載のべき乗剰余演算器は、請求項1記載のべき乗剰余演算器の構成に加えて、第2のレジスタ群は、モンゴメリ乗算剰余演算において、繰り返し累積加算をしていく途中の値を格納する累積加算レジスタと、ダミーレジスタとを含み、モンゴメリ乗算剰余演算実行手段は、モンゴメリ乗算剰余演算とモンゴメリ乗算剰余演算における補正演算の双方に共用して使用される累積加算器を含み、擬似演算実行手段は、補正演算が必要な場合は、累積加算レジスタへの書き込み動作を行い、補正演算が不要な場合は、ダミーレジスタへの書き込み動作を行うためのレジスタ入出力手段を含む。

## 【0053】

請求項9記載のべき乗剰余演算器は、請求項1記載のべき乗剰余演算器の構成に加えて、破棄されるべき中間演算結果を計算する時の演算の入力を与え、前記破棄されるべき中間演算結果を一旦格納するためのダミーレジスタをさらに備える。

## 【0054】

## 【発明の実施の形態】

## 〔実施の形態1〕

（演算の高速化を図るための構成）

まず、電力解析の困難な実施の形態1のべき乗剰余演算器の構成を説明する前提として、べき乗剰余演算器の演算を高速化するための第1および第2の構成を説明しておく。

## 【0055】

## [べき乗剰余演算器の第1のハードウェア構成]

図1を参照して、べき乗剰余演算器の演算を高速化するための第1の構成を有する乗剰余演算器1000は、外部バスとのインタフェースであるI/F（インタフェース）回路101と、鍵eを保持するeレジスタ102と、モンゴメリ変換をする乗数Yを保持するYレジスタ103と、鍵Nを保持するNレジスタ104と、モンゴメリ変換の演算時に行なう $2B+N$ の値を保持する $B2N$ レジスタ105と、平文Xを保持するXレジスタ106と、暗号化および復号化のための演算を行なう演算回路107と、演算結果Pを保持するPレジスタ108と、べき乗剰余演算実行時のステートマシンとしての役割を果たすべき乗剰余制御回路109とを含む。

## 【0056】

べき乗剰余演算回路1000は、さらに、モンゴメリ乗算剰余演算と剰余演算との実行時のステートマシンとしての役割を果たすモンゴメリ乗算剰余・剰余制御回路110と、加算および減算の演算制御を行なう加算・減算制御回路111と、各種モードを保持するモードレジスタ112と、コマンドを保持するコマンドレジスタ113と、ステータスを保持するステータスレジスタ114と、インタフェース回路101、eレジスタ102およびYレジスタ103に接続され、各種レジスタ間でデータのやり取りを行なう内部バス115と、Nレジスタ104、 $B2N$ レジスタ105、Xレジスタ106、演算回路107およびPレジスタ108に接続され、各種レジスタと演算回路107間でデータのやり取りを行なうための内部バス116と、内部バス115および116に接続され、内部バス115および116の間でデータの受け渡しの制御を行なうバス分割回路117とを含む。

## 【0057】

べき乗剰余演算回路1000は、さらに、内部バス115とべき乗剰余制御回路109およびモンゴメリ乗算剰余・剰余制御回路110との間に $E_j/Y_j$ 検出部118を備える。

## 【0058】

### [モンゴメリ演算の演算方法]

べき乗剰余演算を行なうにあたり、高速化を実現するためにモンゴメリ法による乗算剰余演算と剰余演算とを使用しているが、そのうち、式(12)の条件付モンゴメリ演算 [if  $e_j == 1$  then  $Y = MR(X \cdot Y)$ ] の演算方法について説明する。

#### 【0059】

式(12)および(13)のループを実行する際には、eレジスタ102に保持された鍵eが内部バス115上に読み出される。それとともに、べき乗剰余制御回路109は、自身の有するカウンタの状態に従い、鍵eの読み出すべきビットjを $E_j/Y_j$ 検出部118に指示する。 $E_j/Y_j$ 検出部118は、内部バス115上に読み出された鍵eのjビット目の値 $e_j$ を読み込み、べき乗剰余制御回路109に与える。

#### 【0060】

式(20)における $A \bmod 2$ は、乗数Aが式(21)により1ビットずつ右シフトされるため、乗数Aのjビット目に値する。すなわち、Yレジスタ103に保持された乗数Yのjビット目 $Y_j$ である。よって、式(19)～(21)のループを実行する際には、Yレジスタ103に保持された乗数Yが内部バス115上に読み出される。それとともに、モンゴメリ乗算剰余・剰余制御回路110は、自身の有するカウンタの状態に従い、乗数Yの読み出すべきビットjを $E_j/Y_j$ 検出部118に指示する。 $E_j/Y_j$ 検出部118は、内部バス115上に読み出された乗数Yのjビット目の値 $Y_j$ を読み込み、モンゴメリ乗算剰余・剰余制御回路110に与える。

#### 【0061】

剰余演算を行なう際にも同様の処理が行なわれる。

乗算では、バス分割回路117はオフ(OFF)となり、乗数Yは、 $E_j/Y_j$ 検出部118に送られる。自乗算で、被乗数Yは、バス分割回路117をオン(ON)として演算器107に送られ、同時に、乗数として $E_j/Y_j$ 検出部118にも送られる。

#### 【0062】

[べき乗剰余演算器の第2のハードウェア構成]

次に、図1に示した乗剰余演算器1000の構成を変形した乗剰余演算器1100の構成を説明する。

【0063】

図2は、べき乗剰余演算器の演算を高速化するための第2の構成である乗剰余演算器1100の構成を説明する概略ブロック図である。

【0064】

以下、図2の乗剰余演算器1100の構成を図1の乗剰余演算器1000の構成と対比して説明する。

【0065】

図2の乗剰余演算器1100において、eレジスタ102、Yレジスタ103、Xレジスタ106、B2Nレジスタ105、Nレジスタ104、Pレジスタ108およびEj/Yj検出部118は、図1の乗剰余演算器1000の構成要素と同様である。

【0066】

また、乗剰余演算器1100の制御回路100は、図1の乗剰余演算器1000の3つの回路、すなわち、べき乗剰余制御回路109とモンゴメリ乗算剰余・剰余制御回路110と加算・減算制御回路111とを含む。

【0067】

乗剰余演算器1100の0レジスタ130は、値0を保持するレジスタであって、制御回路100からの信号ordにより制御されて、Xレジスタ106、B2Nレジスタ105、Nレジスタ104が選択されないときに、選択されて値が読み出される。なお、自乗算の時は、Xレジスタ106の代わりに、バス分割回路117を介してYレジスタ103が選択される。すなわち、自乗算では、0レジスタ130、Yレジスタ103、B2Nレジスタ105、Nレジスタ104のうちから選択が行われる。以下の説明でも、自乗算の場合は、乗算の場合の処理において、Xレジスタ106とあるのを、Yレジスタ103と読みかえる。

【0068】

乗剰余演算器1100の3つの回路、すなわち、演算器107、1、キャリー

処理部107. 2、右シフト回路107. 3が、図1の乗剰余演算器1000の演算回路107に相当する。

【0069】

レジスタ140は、値0を保持するレジスタであって、制御回路100からの信号0rdにより制御されて、Pレジスタ108が選択されない時、選択されてデータが読み出される。

【0070】

Wr禁止処理部122は、後に説明するif文実行中において、条件式で(ej)が0のとき、Yレジスタ103へのライト信号Ywrをマスクすることにより、Yレジスタ103への書き込みを禁止する。

【0071】

制御回路100からNレジスタ104への補正演算信号は、ORゲート128を介してNレジスタ104に与えられ、補正演算実行時にNレジスタ104の値を読み出すことを指示する信号である。

【0072】

Wr禁止処理部134は、後に説明する補正演算時において、補正が必要でない時にはPレジスタ108へのライト信号Pwrをマスクすることにより、Pレジスタ108への書き込みを禁止する。

【0073】

なお、制御回路100からの信号Xxwrは、Xxレジスタ（Eレジスタ、Yレジスタ、Xレジスタ等）に対する書き込み信号（ライト信号）であり、信号Xxrdは、Xxレジスタに対する読み出し信号（リード信号）である。

【0074】

また、図2においては、外部バス、インタフェース101、モードレジスタ112、コマンドレジスタ113、ステータスレジスタ114等は図示省略されている。

【0075】

乗剰余演算器1100の基本的な動作は、乗剰余演算器1000の動作と同様である。以下、その動作について説明する。

【0076】

図2に示した乗剰余演算器1100の動作は、べき計算は左バイナリ (binary) 法で処理を行い、乗算剰余計算にはモンゴメリ法を用いている。

【0077】

まず、Yレジスタ103に乗法の単位元1をモンゴメリ変換したものを外部、たとえば、CPU (Central Processing Unit) から設定する (ステップS100)。次に、レジスタX106に入力文をモンゴメリ変換したものを外部 (たとえば、CPU) から設定する (ステップS102)。

【0078】

さらに、Nレジスタ104に法を外部 (たとえば、CPU) から設定する (ステップS104)。

【0079】

Rをモンゴメリ変換のための定数 $2^{1024}$ とするとき、次の2行を1024回繰り返す ( $j=1023$  to 0) (ステップS106)。

【0080】

$Y = Y \times Y / R \bmod N$  ... 自乗算

If  $e_j = 1$  then  $Y = X \times Y / R \bmod N$  else void  $X \times Y / R \bmod N$   
... 乗算

続いて、Yの値をモンゴメリ逆変換する ( $Y = 1 \times Y / R \bmod N$ ) (ステップS108)。

【0081】

このYレジスタ103に入ったモンゴメリ逆変換の結果が、べき乗剰余の結果の値となる (ステップS110)。

【0082】

このとき、上述したif文が真か偽かによって、上記乗算結果をYレジスタ103に書くためのライト信号Ywrがマスクされる。

【0083】

また、乗剰余演算器1100の動作では、モンゴメリ乗算剰余の結果Yが、 $N > Y \geq 0$ の範囲に入るように各モンゴメリ乗算剰余の最後で以下の補正演算

if  $P < 0$  then  $P = P + N$

を実行する。このため、乗剰余演算器 1 0 0 0 の動作の場合と異なり、結果をこの範囲にいれるための剰余演算（式（15））は行なわない。

【0084】

（乗剰余演算器 1 1 0 0 のモンゴメリ乗算剰余計算処理）

次に、乗剰余演算器 1 1 0 0 のモンゴメリ乗算剰余計算処理を説明する。ここで、 $Y = X \times Y / R \bmod N$  の場合であって、モンゴメリ乗算剰余処理の最初で  $B = N - X$  を実行する場合について概略説明を行う。

【0085】

まず、Nレジスタ 1 0 4 から法を信号  $Nrd$  で読み出し、Pレジスタ 1 0 8 に信号  $Pwr$  で書き込む（ステップ S 2 0 0）。

【0086】

次に、Xレジスタ 1 0 6 を信号  $Xrd$  で、Pレジスタ 1 0 8 を信号  $Prd$  で読み出し、演算器 1 0 7. 1 で減算（ $P - N$ ）を実行し、右シフトはせずに、信号  $Pwr$  でPレジスタ 1 0 8 に減算結果を書き込む（ステップ S 2 0 2）。

【0087】

さらに、Pレジスタ 1 0 8 を信号  $Prd$  で読み出し、制御回路 1 0 0 からの信号  $Bwr$ （図示せず）でB2Nレジスタ 1 0 5 に減算結果を書き込む（ステップ S 2 0 4）。Pレジスタ 1 0 8 の初期値を 0 とする（ステップ S 2 0 6）。

【0088】

続いて、以下の演算を 1 0 2 4 回繰り返す（ $j = 0$  to 1023）（ステップ S 2 0 8）。

【0089】

i)  $Y_j$  と  $P_0$ 、 $X_0$  で選択したXレジスタ 1 0 6 内の値、Nレジスタ 1 0 4 内の値、B2Nレジスタ 1 0 5 内の値、0レジスタ 1 3 0 内の値を信号  $Xrd$ 、信号  $Nrd$ 、信号  $Brd$  または信号  $Ord$  で読み出す（ステップ S 2 1 0）。

【0090】

ii) 演算器 1 0 7. 1 で加減算を実行し、右シフト回路 1 0 7. 3 で1ビット右ヘシフトした結果を信号  $Pwr$  でPレジスタ 1 0 8 に書きこむ（ステップ S



212)。

【0091】

続いて、以下の補正演算を実行する（ステップS214）。

Nレジスタ104内の値を読み出す。演算器107、1で演算（ $P+N$ ）を実行する。このとき、右シフトは行わない。一つ前のPレジスタ108の値が負であったときは、信号PwrでPレジスタ108に演算（ $P+N$ ）の結果を書きこみ、一方、一つ前のPレジスタ108の値が0または正であった時は、信号Pwrをマスクして、Pレジスタ108に演算（ $P+N$ ）の結果が書きこまれないようにする。

【0092】

以上で、概略説明を終了する。さらに、より具体的に、乗剰余演算器1100の動作を説明すると以下のとおりである。

【0093】

べき乗剰余演算（ $Y = X^e \bmod N$ ）は、例えば、鍵が1024ビットの場合、左バイナリ（binary）法では、以下の処理となる。

【0094】

```

Y=1                      //初期値設定
for j=1023, 0
  Y=Y×Y mod N            //自乗剰余
  If e[j] then Y=X×Y mod N //乗算剰余
End for

```

ここで、 $e[j]$ は $e$ のjビット目の値である。この演算結果はYレジスタ103の値とされる。

【0095】

左バイナリ（binary）法で用いる乗算剰余においては被乗数としてX、またはYを選択するのみで、大部分のハードウェアを変更することなく、以下の2つの演算を選択的に行うことができる。

【0096】

```

Y=Y×Y mod N

```

$$Y = X \times Y \bmod N$$

また、右バイナリ (binary) 法では、 $X = Y^e \bmod N$  は、以下の処理により求めることができる。

【0097】

```

X=1                      //初期値設定
for j=0, 1023
  if e[j] then X=X×Y mod N    //乗算剰余
  Y=Y×Y mod N                //自乗剰余
End for

```

ここで、演算結果はXレジスタ106の値とされる。

【0098】

右バイナリ (binary) 法で用いる乗算剰余においては被乗数の選択に、結果を格納するレジスタの選択する機能を追加するだけで、大部分のハードウェアを変更することなく、以下の2つの演算を選択的に行うことができる。

【0099】

$$Y = Y \times Y \bmod N$$

$$X = X \times Y \bmod N$$

一方、上述したモンゴメリ法 ( $Y = Y \cdot X / R \bmod N$ ) は、同じくYが1024ビット長の場合、以下のような処理になる。

【0100】

```

P=0                      //初期値設定
For j=0 to 1023
  M=(P+Y[j]・X) mod 2=P[0]^Y[j]・X[0]
  P=(P+Y[j]・X+M・N)/2    //累積加算
End for
P=P-N                    ... (22) //減算
If P<0 then P=P+N       //補正演算
Y=P                      //結果のストア

```

図1に示した乗剰余演算器1000では、モンゴメリ法を行うハードウェアの

実装方法として、累積加算の2項目と3項目をあらかじめ算出しておき、3項の加算を2項の加算に変えることで、加算器のハードウェアを減らす構成であった。すなわち、 $B = 2X + N$ を計算している。

【0101】

これに対して、上述した式(22)を含む処理を実装する構成では、

$$B = X + N$$

をあらかじめ計算して、レジスタに記憶しておき、加算器のハードウェアを減らすことができる。

【0102】

すなわち、 $M = P[0] \wedge Y[j] \cdot X[0]$ であるから、MとY[j] (Yのjビット目の値)により以下のようにレジスタを選択して累積加算を実行すればよい。

【0103】

なお、以下では、Pレジスタ108に格納される値を「P」で表し、Xレジスタ106に格納される値を「X」で表し、Nレジスタ104に格納される値を「N」で表し、B2Nレジスタ105に格納される値を「B」で表す。

【0104】

Y[j]	$P[0] \wedge Y[j] \cdot X[0]$	rd信号	累積加算
0	0	0rd	$(P + 0) / 2$
1	0	Xrd	$(P + X) / 2$
0	1	Nrd	$(P + N) / 2$
1	1	Brd	$(P + B) / 2$

上記モンゴメリ法 ( $Y = Y \cdot X / R \bmod N$ ) は、次のように構成して、式(22)の  $P = P - N$  を省くこともできる。(乗剰余演算器1100のモンゴメリ乗算剰余計算処理) 以後の概略説明の内容は、このような処理の例である。

【0105】

$P = 0$  // 初期値設定

For j=0 to 1023

$$M = (P + Y[j] \cdot X) \bmod 2 = P[0] \wedge Y[j] \cdot X[0]$$

$P = (P + Y[j] \cdot X - M \cdot N) / 2$  // 累積加減算

End for

If  $P < 0$  then  $P = P + N$  // 補正演算

$Y = P$  // 結果のストア

この場合、加減算の2項目と3項目について、 $B = N - X$ をあらかじめ計算しレジスタに記憶しておき、加減算において、以下の計算を行うこととしてもよい。

【0106】

$P = (P - (N - X)) / 2$

あるいは、 $B = X - N$ を予めレジスタに記憶しておき、加減算において、以下の計算を行うこととしてもよい。

【0107】

$P = (P + (X - N)) / 2$

上記いずれの方法によっても、加算器のハードウェア量を減らすことができる。すなわち、 $M = P[0] \wedge Y[j] \cdot X[0]$ であるから、 $M$ と $Y[j]$ により以下のようにレジスタの選択と加算または減算して累積加減算を実行すればよい。

【0108】

Y[j]	M	+/-	rd信号	累積加算
0	0	+	0rd	$(P + 0) / 2$
1	0	+	Xrd	$(P + X) / 2$
0	1	-	Nrd	$(P - N) / 2$
1	1	-	Brd	$(P - B) / 2$

ここで、 $B = N - X$ である。

【0109】

モンゴメリ法 ( $Y = Y \cdot X / R \bmod N$ ) の実行は $X$ を $Y$ におきかえればよい。すなわち $X$ レジスタ106に対する $Xrd$ 信号の行き先を $Y$ レジスタ103に切り替え、バス分割回路117を介して読み出すだけで処理を実行できる。

【0110】

また累積加減算中の“/2”は右シフト処理で行う。補正演算についても、バイナリ (binary) 法と同様に、書き込み信号 (ライト信号) をマスクすることで実行時間を入力データによらず一定にすることが、次のようにすれば可能である。

【0111】

if  $P < 0$  then  $P = P + N$  else void  $P + N$

すなわち、補正演算時はNレジスタ104にNrd信号を与え、補正要否の信号によりPレジスタ108へのライト信号Pwrをマスクすればよい。

【0112】

なお、図2において、M、YjによるXxrd信号の選択や、初期値 $P = 0$ の設定、Bの値を記憶するレジスタへの値の算出と設定等は制御回路100内で適切な制御信号を発生し、演算器に与えることで処理している。図2において、べき乗剰余の初期値は、たとえば、CPUから与えられる。

【0113】

〔実施の形態1の乗剰余演算器1200の構成〕

以上の準備の下に、図2に示した乗剰余演算器1100の構成を基礎として、さらに電力解析が困難なように構成された乗剰余演算器1200について、以下に説明する。

【0114】

図3は、実施の形態1の乗剰余演算器1200の構成を説明するための概略ブロックである。

【0115】

乗剰余演算器1200では、以下に説明するように、図2の乗剰余演算器1100にKレジスタ132を追加して、べき乗剰余演算中の

if  $e_j = 1$   $Y = X \times Y / R \bmod N$  else void  $X \times Y / R \bmod N$

との処理の代わりに、

if  $e_j = 1$   $Y = X \times Y / R \bmod N$  else  $K = X \times Y / R \bmod N$

との処理を行う。

【0116】

図 3 では、左バイナリ (binary) 法の場合の構成を例として示している。

図 3 を参照して、追加した K レジスタ 1 3 2 の動作についてさらに詳しく説明する。

【0 1 1 7】

べき乗剰余演算のバイナリ法で、上述した if 文の実行において、if 文内の判断が“偽”の場合に、結果を書き込むレジスタとして K レジスタ 1 3 2 を用いる。

【0 1 1 8】

なお、 $e[j]$  の値は、Y の自乗演算中に確認し、if 文の処理の前にその値を確定しておく。

【0 1 1 9】

ここで、 $Y = X^e \bmod N$  の処理を、1 0 2 4 ビットのデータに対して行う場合は、以下のとおりとなる。

【0 1 2 0】

$Y = 1$

For  $j = 1023$  to 0

$Y = Y \cdot Y \bmod N$

If ( $e[j] = 1$ ) then  $Y = Y \cdot X \bmod N$  else  $K = Y \cdot X \bmod N$

End for

上述の処理でモンゴメリ法を用いた場合は、以下のようになる。

【0 1 2 1】

$X = X \cdot R \bmod N$

$Y = R \bmod N$

For  $j = 1023$  to 0

$Y = Y \cdot Y / R \bmod N$

If ( $e[j] = 1$ ) then  $Y = Y \cdot X / R \bmod N$  else  $K = Y \cdot X / R \bmod N$

End for

$Y = Y / R \bmod N$

ここで、Y レジスタ 1 0 3 への書き込み信号  $Y_{wr}$  をマスクするために、 $E_j / Y_j$  検出部 1 1 8 からの信号  $e[j]$  を用いる。

## 【0122】

if文が実行される際に、信号 $e[j]$ に応じてWr禁止処理部122から出力される信号（信号 $e[j]$ の反転信号）を反転した信号と制御回路100からのYレジスタ103への書き込み信号Ywrとの論理積をゲート回路124で演算した結果が、Yレジスタ103に書き込み信号Ywr'として与えられる。

## 【0123】

一方、信号 $e[j]$ に応じてWr禁止処理部122から出力される信号（信号 $e[j]$ の反転信号）とYレジスタ103への書き込み信号Ywrとの論理積をゲート回路126で演算した結果が、Kレジスタ132に書き込み信号Kwrとして与えられる。

## 【0124】

なお、右バイナリ（binary）法でも同様な処理を行うことが可能である。

以上のように乗剰余演算器1200を構成することによりif文の条件が“真”であっても、また“偽”であっても同様にレジスタへの書き込み処理が行われることになる。このため、いずれの場合にも、書き込み時の電流が流れることとなり、電力解析に対する耐性が向上する。

## 【0125】

## 〔実施の形態2〕

図4は、本発明の実施の形態2の乗剰余演算器1300の構成を説明するための概略ブロック図である。

## 【0126】

以下に説明するとおり、B2Nレジスタ105を、中間データBを書き込むためのレジスタとして用いて、べき乗剰余演算中の

if  $e_j = 1$   $Y = X \times Y / R \bmod N$  else void  $X \times Y / R \bmod N$

との処理の代わりに、

if  $e_j = 1$   $Y = X \times Y / R \bmod N$  else  $B = X \times Y / R \bmod N$

との処理を行う。

## 【0127】

すなわち、if文において、その論理が“偽”の場合に、ダミー処理としてデ-

タの書き込みを行うレジスタを別途設けるのではなく、B2Nレジスタ105をこのようなダミー処理（中間データBの書き込み）を行うレジスタとしても共用する。

【0128】

なお、図4においては、図3のWr禁止処理部134とゲート回路138の代わりに、補正演算の要否に基づく信号を出力する補正要否判定部150とゲート回路138とを用いている。

【0129】

B2Nレジスタ105はモンゴメリ乗算剰余を演算する時のテンポラリレジスタであり、if文の論理が“偽”の場合に、データを書き込んでも問題はない。

【0130】

つまり、 $Y = X^e \bmod N$ の処理を、1024ビットのデータに対して行う場合は、以下のとおりとなる。なお、B2Nレジスタ105のデータの値をB2Nとする。

【0131】

$Y = 1$

For  $j = 1023$  to 0

$Y = Y \cdot Y \bmod N$

If ( $e[j] = 1$ ) then  $Y = Y \cdot X \bmod N$  else B2N =  $Y \cdot X \bmod N$

End for

上述の処理でモンゴメリ法を用いた場合は、以下のようなになる。

【0132】

$X = X \cdot R \bmod N$

$Y = R \bmod N$

For  $j = 1023$  to 0

$Y = Y \cdot Y / R \bmod N$

If ( $e[j] = 1$ ) then  $Y = Y \cdot X / R \bmod N$  else B2N =  $Y \cdot X / R \bmod$

N

End for



$$Y = Y / R \bmod N$$

ここで、Yレジスタ103への書き込み信号Ywrをマスクするために、Ej/Yj検出部118からの信号e[j]を用いる。

#### 【0133】

if文が実行される際に、信号e[j]に応じてWr禁止処理部122から出力される信号（信号e[j]の反転信号）を反転した信号と制御回路100からのYレジスタ103への書き込み信号Ywrとの論理積をゲート回路124で演算した結果が、Yレジスタ103に書き込み信号Ywr'として与えられる。

#### 【0134】

一方、信号e[j]に応じてWr禁止処理部122から出力される信号（信号e[j]の反転信号）とYレジスタ103への書き込み信号Ywrとの論理積をゲート回路126で演算した結果が、B2Nレジスタ105に書き込み信号B2Nwrとして与えられる。

#### 【0135】

なお、右バイナリ（binary）法でも同様な処理を行うことが可能である。

以上のように乗剰余演算器1200を構成することによりif文の条件が“真”であっても、また“偽”であっても同様にレジスタへの書き込み処理が行われることになる。このため、いずれの場合にも、書き込み時の電流が流れることとなり、電力解析に対する耐性が向上する。

#### 【0136】

##### 〔実施の形態3〕

図5は、本発明の実施の形態3の乗剰余演算器1400の構成を説明するための概略ブロック図である。

#### 【0137】

以下、図5に示した乗剰余演算器1400の動作を図2に示した乗剰余演算器1100の動作と対比させて説明する。

#### 【0138】

まず、図5に示した乗剰余演算器1400の動作では、図2に示した乗剰余演算器1100の動作における、べき乗剰余中の処理のうち、

「Yレジスタ103に乗法の単位元1をモンゴメリ変換したものを外部、たとえば、CPUから設定する（ステップS100）。」

との処理を省略する。

【0139】

さらに、図2に示した乗剰余演算器1100の動作における補正演算の動作のうち、ステップS214の

「Nレジスタ104内の値を読み出す。演算器107、1で演算（ $P+N$ ）を実行する。このとき、右シフトは行わない。一つ前のPレジスタ108の値が負であったときは、信号PwrでPレジスタ108に演算（ $P+N$ ）の結果を書きこみ、一方、一つ前のPレジスタ108の値が0または正であった時は、信号Pwrをマスクして、Pレジスタ108に演算（ $P+N$ ）の結果が書きこまれないようにする。」

との処理に以下の内容を追加する。

【0140】

「ただし、初めて $e[j] = 1$ となったときは、Nレジスタ104とPレジスタ108の代わりに、Xレジスタ106と0レジスタ140からデータ読み出す。演算器107で演算（ $0+X$ ）を実行する。一つ前のPレジスタ108の値の正負に係らず、書き込み信号PwrでPレジスタに演算結果（ $0+X$ ）を書き込む。」

すなわち、実施の形態3の乗剰余演算器1400では、以下の問題を回避することを目的としている。

【0141】

つまり、べき乗剰余演算で右バイナリ法を用いた場合、Yの初期値は1であり $e[j]$ が1になるまで、Yレジスタ103中の値は変化しない。そして、一旦、 $e[j] = 1$ となり $Y = XY$ となった後は、Yレジスタ103中の値は、入力文Xに依存した値となる。したがって、入力文を換えて、どのタイミングで入力文に依存した電流になるかをみれば、 $e[j]$ の上位ビットから何ビットまで0が続いているかが外部から判断できてしまうことになる。

【0142】

そこで、実施の形態3の乗剰余演算器1400では、大略、以下のような処理

を行う。

#### 【0143】

すなわち、初回の $e[j] = 1$ との条件が満たされたときは、 $YX$ の結果は $X$ となる。つまり、それ以前の $Y$ の値にかかわらず、初回に $e[j] = 1$ となった時点で $Y = X$ とする処理を行いさえすれば、最終的には同じ結果が得られる。このような手法を用いた場合、初回に $e[j] = 1$ となる以前の $Y$ の値が無関係となるため、 $Y$ の初期値を入力文に依存した値に設定できる。このため、 $Y$ のビットの全期間にわたって、電流を入力文に依存するようにし、0の個数を分からなくすることができる。

#### 【0144】

図5に示すとおり、実施の形態3の乗剰余演算器1400では、初回の $e[j] = 1$ を検出するための初回 $E_j$ 値検出部161が設けられ、検出結果により乗算結果を $X$ そのものに変更できる乗算回路が用いられている。

#### 【0145】

図5では、乗算回路としてモンゴメリ乗算剰余回路を用いた例が示されている。

#### 【0146】

ここで、図1に示した乗剰余演算器1000と乗剰余演算器1400との相違をまとめると以下のとおりである。

乗剰余演算器1000では、補正演算を最後にまとめて剰余をとることで行っているのに対し、乗剰余演算器1400では各モンゴメリ乗算剰余内の最後に行うようにしている。また、 $B = N - X$ を各モンゴメリ乗算剰余の最初に計算し、 $B$ レジスタ105を演算器107の入力として選択した時、または $N$ レジスタ104を選択した時には、演算器107で減算を実行する。このことにより補正対象となる結果が、 $(-N)$ から $(N-1)$ の範囲となるため、補正演算の実行の要否を演算器107中のキャリー判定部107.2で判定できる。

#### 【0147】

実施の形態3の乗剰余演算器1400の処理をさらに詳しく説明する。

まず、初回 $E_j$ 値検出部161の動作を中心に説明する。

## 【0148】

乗剰余演算器1400でのべき乗剰余では、次のようにSレジスタ160を用いる。Sレジスタ160は初期値を0とし、ループが1回終わる毎に、 $EJ (=e[j])$  が1であった時、1を記憶する。

## 【0149】

S=0

For j=1023 to 0

Y = Y · Y mod N、EJ=e[j]

If (EJ=1) then if (S=1) then Y = Y · X mod N else Y = X

if EJ=1 then S=1

End for

上記処理において、モンゴメリ法を用いた場合は、以下のとおりとなる。

## 【0150】

X = X · R mod N

S=0

For j=1023 to 0

Y = Y · Y / R mod N

If (EJ=1) then if (S=1) then Y = Y · X / R mod N else Y = X

if EJ=1 then S=1

End for

Y = Y / R mod N

上述の処理のとおり、実施の形態3の乗剰余演算器1400においては、初回の $e[j]=1$ についての検出信号(L:EJに相当)により乗算結果がXに設定される。

## 【0151】

モンゴメリ演算の最後の補正演算において、初回の $e[j]=1$ の検出信号Lにより、次のように、演算器107の入力として、Pレジスタ108の値の代わりに0レジスタ130からの値0を選択し、また、Nレジスタ104や0レジスタ130の値の代わりにXレジスタ106中の値を選択することで、 $P=X$ とするこ

とが出来る。その後、Pレジスタ108中の値をYレジスタ103に格納するためのデータの転送を行う。

#### 【0152】

／／補正演算

キャリー	L	計算
1	0	$P = P + N$
0	0	$P = P + 0$
X	1	$P = 0 + X$

その後に、 $Y = P$ とする処理を行うことで、正に補正された乗算剰余結果またはXの値がYレジスタ103に記憶される。

#### 【0153】

ここで、Yレジスタ103に格納される値Yの初期値は、最終結果に影響しないためDPAを考慮し、種々の設定方法を取ることができる。たとえば、それ以前の処理で、Yレジスタ103に残っている値をそのまま使用することも可能である。

#### 【0154】

また、例えば、リダクション (reduction) 後の結果をYレジスタ103に書きこみ、モンゴメリ変換後の結果をXレジスタ106に書きこみ、そのままバイナリ法を実行する。すると、 $Y = Y \times Y$ は、 $(M \bmod N) \times (MR \bmod N) / R \bmod N$ となるので、 $M \times M \bmod N$ となるため、Rが抜け落ちており異なる入力となる。このため、バイナリ法で一致するパターンはまれにしか現れない。

#### 【0155】

以上のような構成では、最初の1が現れたか否かを示すデータを保持するSレジスタ160を設けられる。それにより、最初の1が現れた際に、モンゴメリ演算の補正演算中に初期値を設定することにより単位元を最初に設定する必要がなくなる。このため、入力データの依存の有無が現れる電流消費のパターンとなって最初に0が連続する個数が見破られることを困難にできる。このような処理は、モンゴメリの補正演算のオペランドを切り替えるだけで実現できるため、電流への影響がほとんどなく、実行サイクルも変わらない。

〔実施の形態 4〕

図 6 は、本発明の実施の形態 4 の乗剰余演算器 1 5 0 0 の構成を説明するための概略ブロック図である。

【0 1 5 6】

以下、図 6 に示した乗剰余演算器 1 5 0 0 の動作を、図 2 に示した乗剰余演算器 1 1 0 0 の動作と対比させて説明する。

【0 1 5 7】

まず、図 6 に示した乗剰余演算器 1 5 0 0 の動作では、図 2 に示した乗剰余演算器 1 1 0 0 の動作における、モンゴメリ乗算剰余中の

「P レジスタ 1 0 8 の初期値を 0 とする（ステップ S 2 0 6）。」

との処理を省略する。

【0 1 5 8】

また、図 2 に示した乗剰余演算器 1 1 0 0 のモンゴメリ乗算剰余中の

「続いて、以下の演算を 1 0 2 4 回繰り返す（j=0 to 1023）（ステップ S 2 0 8）。」

【0 1 5 9】

i) Y<sub>j</sub> と P<sub>0</sub>、X<sub>0</sub> で選択した X レジスタ 1 0 6 内の値、N レジスタ 1 0 4 内の値、B<sub>2</sub>N レジスタ 1 0 5 内の値、0 レジスタ 1 3 0 内の値を信号 X<sub>rd</sub>、信号 N<sub>rd</sub>、信号 B<sub>rd</sub> または信号 0<sub>rd</sub> で読み出す（ステップ S 2 1 0）。

【0 1 6 0】

i i) 演算器 1 0 7. 1 で加減算を実行し、右シフト回路 1 0 7. 3 で 1 ビット右へシフトした結果を信号 P<sub>wr</sub> で P レジスタ 1 0 8 に書きこむ（ステップ S 2 1 2）。」

との処理では、P レジスタ 1 0 8 を読み出し、その値に加減算を行うのに対し、図 6 に示した乗剰余演算器 1 5 0 0 では、

「1 0 2 4 回の中で初めて Y<sub>j</sub> = 1 である時に、P レジスタ 1 0 8 の代わりに、0 レジスタ 1 4 0 の値を読み出す。」

ように処理を変更する。

【0 1 6 1】

すなわち、実施の形態4の乗剰余演算器1500では、以下の問題を回避することを目的としている。

#### 【0162】

つまり、モンゴメリ乗算剰余、または乗算の場合、Pレジスタ108内のデータを右シフトしながら、 $Y[j] = 1$ の時Xを加算していく。その際、Pの初期値は0としている。そのため、 $Y[j]$ が1になるまでPレジスタ108内の値は変化しない。一旦、 $Y[j] = 1$ となった後は、繰り返しの中に右シフトを含むため、Pレジスタ108に書きこむ値は繰り返し毎に異なった値となる。したがって、入力文を換えて、どのタイミングで入力文に依存した電流になるかをみれば、それぞれの入力文毎に、 $Y[j]$ の下位ビットから何ビット分0が続いているか判断できることになってしまう。

#### 【0163】

そこで、実施の形態4の乗剰余演算器1500では、大略、以下のような処理を行う。

#### 【0164】

初回の $Y[j] = 1$ でPレジスタに入っている値は0である。すなわち、それ以前のPの値にかかわらず、その時点で0をPレジスタ108から読み出すようにすれば、最終的には同じ結果が得られる。このような手法を用いた場合、 $Y[j] = 1$ となる以前のPレジスタ108内の値が最終的な結果とは無関係となる。このため、Pレジスタ108の初期値を入力文に依存した値に設定でき、 $Y[j]$ のビットの全期間、電流を入力文に依存するようにし、0の個数を外部から電力解析によって分からなくすることができる。

#### 【0165】

図6に示すとおり、実施の形態4の乗剰余演算器1500では、初回の $Y[j] = 1$ を検出するための初回 $Y[j]$ 検出部180と、検出結果によりPレジスタ108からデータを読み出す代わりに、0レジスタ140から0を読み出すよう変更する構成が設けられている。

#### 【0166】

図6の実施の形態4の乗剰余演算器1500は、モンゴメリ乗算剰余回路であ

る。

【0167】

図6に示すとおり、乗剰余演算器1500において、初回Y[j]検出部180の出力をHとする。

【0168】

べき乗剰余の処理では、次のようにTレジスタ182を用いる。Tレジスタ182は初期値を0とし、処理ループが1回終わる毎に、Y[j]が1であった時、1を記憶する。出力Hは、(T=0)かつ(Y[j]=1)であるときに“1”となる。

【0169】

T=0

For j=0 to 1023

    If H=0 then

        M = (P + Y[j] · X) mod 2

        P = (P + Y[j] · X + M · N) / 2

    Else

        M = (0 + Y[j] · X) mod 2

        P = (0 + Y[j] · X + M · N) / 2

    If Y[j] = 1 then T=1

End for

図6に示すように、モンゴメリ演算中において、初回のY[j]=1を検出したことを示す検出信号Hをゲート回路172およびゲート回路174の一方入力に与える。これにより、次のように、演算器107の入力として、Pレジスタ108の値の代わりに0レジスタ140の値を選択し、P=Xとすることができる。

【0170】

／／モンゴメリ演算中

H

計算内容

0

M = (P + Y[j] · X) mod 2



$$0 \quad P = (P + Y[j] \cdot X + M \cdot N) / 2$$

$$1 \quad M = (0 + Y[j] \cdot X) \bmod 2$$

$$1 \quad P = (0 + Y[j] \cdot X + M \cdot N) / 2$$

また、Pレジスタ108に対する初期値の例としては、例えば、モンゴメリ乗算剰余において、Pレジスタ108を初期化せず、ひとつ前の演算結果をそのまま用いることが可能である。

#### 【0171】

以上のような構成とすれば、乗算、モンゴメリ乗算剰余において、Y[j]をLSB (Least Significant Bit) から演算する場合、最初の1が現れるまで消費電流の入力文からの依存性が少なくなることを防ぐことができる。これにより、電流変動パターンからPレジスタ108に格納されるべき値Pには、LSBから連続する0が何個あるかが見破られることを困難にすることが可能である。

#### 【0172】

##### 【実施の形態5】

図7は、本発明の実施の形態5の乗剰余演算器1600の構成を説明するための概略ブロック図である。

#### 【0173】

以下、図7に示した乗剰余演算器1600の動作を、図2に示した乗剰余演算器1100の動作と対比させて説明する。

#### 【0174】

まず、図7に示した乗剰余演算器1600の動作では、図2に示した乗剰余演算器1100の動作における、補正演算中のステップS214の

「続いて、以下の補正演算を実行する（ステップS214）。

#### 【0175】

Nレジスタ104内の値を読み出す。演算器107.1で演算(P+N)を実行する。このとき、右シフトは行わない。一つ前のPレジスタ108の値が負であったときは、信号PwrでPレジスタ108に演算(P+N)の結果を書きこみ、一方、一つ前のPレジスタ108の値が0または正であった時は、信号Pwrをマスクして、Pレジスタ108に演算(P+N)の結果が書きこまれないよ

うにする。」

との処理を以下のように変更する。

【0176】

「続いて、以下の補正演算を実行する（ステップS214'）。

ひとつ前のPが負であった時は、Nレジスタ104からデータを読み出し、ひとつ前のPが0または正であった時は、0レジスタ130から0を読み出す。演算結果を書き込み信号PwrをマスクせずにPレジスタ108に書き込む。」

すなわち、実施の形態5の乗剰余演算器1600では、以下の問題を回避することを目的としている。

【0177】

つまり、ハードウェアの追加を抑制して、補正演算を実現するためには、演算器107、Pレジスタ108等を共用することが通例である。この場合、補正前の値がPレジスタ108に入っているため、補正が必要な時にのみ補正演算を実行し、Pレジスタ108に書きこめばよい。あるいは、補正演算実行の有無による時間変動を避けるため補正の要否にかかわらず補正演算は実行し、補正が不要である時、Pレジスタへの書き込み信号をマスクする方法を採用することもできる。しかしながら、このように構成した場合、Pレジスタへの書きこみ信号の有無により消費電流に差が生じてしまう。

【0178】

図7に示した乗剰余演算器1600は、実施の形態2の乗剰余演算器1300の構成を変更して、Nレジスタ104および0レジスタ130の読出しを制御する信号Nr dおよび信号0 r dをそれぞれORゲート184および186の一方入力に与える。ORゲート184および186の他方入力には、補正の要否を判定する補正要否判定部150からの出力が与えられる。また、乗剰余演算器1600では、補正の要否に応じて、Pレジスタ108への書き込みを制御するゲート回路138'がない。

【0179】

すなわち、補正演算時、Pレジスタ108中の値Pの正負、すなわち加減算を行う演算器107、1のキャリーに基づいて、キャリーが負の時は、Nレジスタ

を選択して演算 ( $P+N$ ) を実行する。一方、キャリーが正の時は、0レジスタ130を選択して演算 ( $P+0$ ) を実行する。これによりいずれの場合でも、演算結果がPレジスタ108に書き込まれる処理が行われることとなり、書き込みの有無による消費電流の差を回避できる。

【0180】

そして、Pレジスタ108から読み出されたデータを、Yレジスタ103に格納する。

【0181】

このような構成とすることで、補正が不要の場合も、結果をレジスタに書き込むという動作が必ず行われるので、レジスタへの書き込みの有無より消費電流に差が生じることを回避できる。このため、電力解析により外部から処理を推定することが困難となる。

【0182】

〔実施の形態6〕

図8は、本発明の実施の形態6の乗剰余演算器1700の構成を説明するための概略ブロック図である。

【0183】

以下、図8に示した乗剰余演算器1700の動作を、図7に示した乗剰余演算器1600の動作と対比させて説明する。

【0184】

図8に示した乗剰余演算器1700の動作では、図7に示した乗剰余演算器1600の動作における、補正演算中のステップS214'の

「続いて、以下の補正演算を実行する（ステップS214'）。

【0185】

ひとつ前のPが負であった時は、Nレジスタ104からデータを読み出し、ひとつ前のPが0または正であった時は、0レジスタ130から0を読み出す。演算結果を書き込み信号PwrをマスクせずにPレジスタ108に書き込む。」

との処理を以下のように変更する。

【0186】

「続いて、以下の補正演算を実行する（ステップ S214」）。

ひとつ前の P が負であった時は、N レジスタ 104 からデータを読み出し、ひとつ前の P が 0 または正であった時は、0 レジスタ 130 から 0 を読み出す。演算結果を右シフトし書き込み信号 Pwr をマスクせずに P レジスタ 108 に書き込む。

#### 【0187】

その結果を P レジスタ 108 から読み出したとき、左シフトして元に戻す。」

すなわち、実施の形態 6 の乗剰余演算器 1700 では、以下の問題を回避することを目的としている。

#### 【0188】

つまり、実施の形態 5 の乗剰余演算器 1600 では、P レジスタ 108 に、P に格納されていたのと同じ値 ( $P+0$ )、または異なる値 ( $P+N$ ) を書くことになるから電流にわずかに差が残る可能性がある。このため、電力解析の余地が残ってしまう可能性がある。

#### 【0189】

このような問題を解消するために、実施の形態 6 の乗剰余演算器 1700 では、補正演算では加減算の後、右シフト処理を行う必要はないものの、あえて右シフト処理を行うことで、補正演算完了後の P レジスタ 108 に書き込む値を、補正が否であっても異なるようにする。これにより、補正の要否による消費電流の差をさらに少なくする。

#### 【0190】

右シフトを行った時にあふれでた LSB は、別途 V レジスタ 190 を設けて保存しておき、P レジスタ 108 の値を他のレジスタに転送する場合に、左シフト処理の実行と V レジスタ 190 に記憶しておいた LSB 値を入れることで、正しい値に戻す。

#### 【0191】

図 8 に示した乗剰余演算器 1700 では、実施の形態 5 の乗剰余演算器 1600 の構成に加えて、演算器 107 の処理の後に行われる右シフト処理時のキャリーを記憶し、P レジスタ 108 からのデータの読出しの後に、読み出されたデー

タに対して行われる左シフト処理に対して記憶したキャリーを与えるVレジスタ190を設ける。右シフト回路107、3は、制御回路100から補正演算が指示された際および右シフト処理が指示された際に、活性状態となる信号を出力するOR回路188により制御される。さらに、乗剰余演算器1700では、上述した左シフト処理を行うための左シフト回路192が設けられる。

#### 【0192】

補正演算を実行し、LSBをVレジスタ190に書き込み、右シフト処理を行うことで、Pレジスタ108には、もともとPレジスタ108に記憶されていたのとは異なる値（右シフトした値）を記憶する。

#### 【0193】

補正演算実行後、Pレジスタ108からデータ読み出す際には、左シフト回路192で左シフトを実行し、LSBにVレジスタ190の値を入力する。その値をYレジスタ103に格納する。

#### 【0194】

以上のような構成とすることで、補正演算においてももともとPレジスタ108に格納されていたのと同じ値を再度Pレジスタ108に書き込むという動作をなくすことができ、Pレジスタ108への書き込み処理において、補正の有無に応じて電流が増減することを回避できる。

#### 【0195】

##### 〔実施の形態7〕

図9は、本発明の実施の形態7の乗剰余演算器1800の構成を説明するための概略ブロック図である。

#### 【0196】

以下、図9に示した乗剰余演算器1800の動作を、図2に示した乗剰余演算器1100の動作と対比させて説明する。

#### 【0197】

まず、図9に示した乗剰余演算器1800の動作では、図2に示した乗剰余演算器1100の動作における、補正演算中のステップS214の

「続いて、以下の補正演算を実行する（ステップS214）。

## 【0198】

Nレジスタ104内の値を読み出す。演算器107.1で演算( $P+N$ )を実行する。このとき、右シフトは行わない。一つ前のPレジスタ108の値が負であったときは、信号PwrでPレジスタ108に演算( $P+N$ )の結果を書きこみ、一方、一つ前のPレジスタ108の値が0または正であった時は、信号Pwrをマスクして、Pレジスタ108に演算( $P+N$ )の結果が書きこまれないようにする。」

との処理を以下のように変更する。

## 【0199】

「続いて、以下の補正演算を実行する(ステップS214)。

Nレジスタ104内の値を読み出す。演算器107.1で演算( $P+N$ )を実行する。このとき、右シフトは行わない。一つ前のPレジスタ108の値が負であったときは、信号PwrでPレジスタ108に演算( $P+N$ )の結果を書きこみ、一方、一つ前のPレジスタ108の値が0または正であった時は、別のQレジスタ194に演算( $P+N$ )の結果を書きこむ。」

すなわち、実施の形態7の乗剰余演算器1800でも、実施の形態6の乗剰余演算器1700で回避しようとするのと同じ問題を回避することを目的とする。

## 【0200】

このような問題を解消するために、実施の形態7の乗剰余演算器1800では、実施の形態1の乗剰余演算器1200の構成に加えて、Qレジスタ194とWr禁止処理部134と書き込み信号Pwrを受けて論理積をとりQレジスタ194への書き込みを制御する信号を生成するゲート回路196とが設けられる。

## 【0201】

このような構成では、補正演算の要否にかかわらず補正演算( $P+N$ )そのものは常に実行し、否の時にも結果の書き込みを行うためのQレジスタ194が設けられることになる。Qレジスタ194へは、補正の要否に応じて、演算結果( $P+N$ )が、Pレジスタ108に対して相互排他的に書き込まれる。

## 【0202】

以上のような構成において、計算値( $P+N$ )の値は、通常は、Qレジスタ1

94にもともと格納されている値ともPレジスタ108にもともと格納されている値とも異なる。このため、Pレジスタ108に書き込む値が、もともとPレジスタ108に格納されている値と同じか否かによる電流の差を少なくできる。

#### 【0203】

##### 〔実施の形態8〕

図10は、本発明の実施の形態8の乗剰余演算器1900の構成を説明するための概略ブロック図である。

#### 【0204】

実施の形態8の乗剰余演算器1900の目的は以下のとおりである。すなわち、左バイナリ (binary) の場合、 $e[j] = 1$ の乗算と次の自乗算で乗数であるYの値は異なるが、 $e[j] = 0$ の時は同じ値となる。また右バイナリ (binary) の場合、 $e[j] = 1$ の乗算と次の乗算で被乗数であるXの値は異なるが、 $e[j] = 0$ の時は同じ値となる。このため $e[j] = 0$ の時だけ消費電流に同じパターンが含まれ電流解析に弱くなる可能性が残る。実施の形態8の乗剰余演算器1900は、このようなおそれを解消する。

#### 【0205】

図10に示した実施の形態8の乗剰余演算器1900は、実施の形態1の乗剰余演算器1200の構成を上述した右バイナリ (binary) 法に適應するように変更し、さらに、if文の判断が"false"の時の被乗数をXでなく、Kレジスタ132中の値としたものである。

#### 【0206】

このために、乗剰余演算器1900においては、乗剰余演算器1200と比べて、ゲート回路124が省略されて、信号YwrはそのままYレジスタ103に与えられる。さらに、Wr禁止処理部122からの信号の反転レベルと信号Xwrの論理積をXレジスタ106に与えるゲート回路210と、Wr禁止処理部122からの信号の反転レベルと信号Xrdの論理積をXレジスタ106に与えるゲート回路212とが設けられる。

#### 【0207】

さらに、ゲート回路126は、信号Ywrの代わりに信号Xrdを受ける。

また、乗剰余演算器 1900 においては、Wr 禁止処理部 122 からの信号と信号 Xrd の論理積を K レジスタ 132 に与えるゲート回路 214 が設けられる。

#### 【0208】

その他の構成は、実施の形態 1 の乗剰余演算器 1200 の構成と同様であるので、同一部分には、同一符号を付してその説明は繰り返さない。

#### 【0209】

この乗剰余演算器 1900 の処理では、バイナリ (binary) 法での繰り返し部分は、次の通りとなる。

#### 【0210】

```
for j=0 to 1023
  if e[j]=1 then X=X×Y/R mod N else K=K×Y/R mod N
  Y=Y×Y/R mod N
end for
```

乗剰余演算器 1900 で信号 Ywr は自乗算の結果を Y レジスタ 103 に書き込むための信号である。また、信号 Xwr は、 $e[j]=1$  の時、乗算結果を X レジスタ 106 に書き込み、 $e[j]=0$  のとき K レジスタ 132 に書き込むための信号である。さらに、信号 Xrd は、 $e[j]=1$  の時、乗数を X レジスタ 106 から読み出し、 $e[j]=0$  の時、K レジスタ 132 から読み出すための信号である。

#### 【0211】

以上のような構成とすることで、乗剰余演算器 1900 の処理では、Y レジスタ 103 の値は  $e[j]$  に関わらず、常に自乗算でのみ変化する。

#### 【0212】

$e[j]=1$  で X を乗数とする時は、X は必ず変化し、 $e[j]=0$  で K を乗数とした場合も K は必ず変化する。

#### 【0213】

このため、乗数の値は繰り返し毎に異なることとなり、乗算実行時の消費電流の  $e[j]$  との相関を一層低く抑えることが可能となる。

#### 【0214】



なお、左バイナリ法の場合は以下のとおりとなる。

左バイナリ (binary) 法の場合には、繰り返し部分は

```
for j=1023 to 0
```

```
Y=Y×Y/R mod N
```

```
if e[j]=1 then Y=X×Y/R mod N else J=X×J/R mod N
```

```
end for
```

となる。Jは被乗数のダミーとして設けたレジスタ中に格納される値である。効果は右バイナリ法の場合と同じである。また、その構成を示す図および説明は図 10 に対して、上述のようなダミーレジスタ J を設けることで実現できるので、省略する。

#### 【0215】

なお、以上、実施の形態 1～8 において、それぞれの目的に対応するための乗剰余演算器の構成をそれぞれ説明したが、これらの実施の形態の構成を組み合わせた構成により、2 以上の目的に同時に対処することが可能な乗剰余演算器を実現することができる。また、右バイナリ (Binary) 法、左バイナリ (Binary) 法のいずれに対しても適用することが可能である。

#### 【0216】

今回開示された実施の形態はすべての点で例示であって制限的なものではないと考えられるべきである。本発明の範囲は上記した説明ではなくて特許請求の範囲によって示され、特許請求の範囲と均等の意味および範囲内でのすべての変更が含まれることが意図される。

#### 【0217】

##### 【発明の効果】

請求項 1 記載のべき乗剰余演算器では、回路規模が小さく、高速処理が可能なべき乗剰余演算器を提供することが可能となる。しかも、モンゴメリ乗算剰余演算およびべき乗剰余演算の各計算結果を得るためには省略可能な中間演算処理も擬似的に実行されるので、電力解析に対する耐性が向上する。

#### 【0218】

請求項 2 および 3 記載のべき乗剰余演算器では、破棄されるデータが計算途中

に現れた場合でもレジスタへの書き込みが行われるので、書き込み時の電流が流れることとなり、電力解析に対する耐性が向上する。

#### 【0219】

請求項4記載のべき乗剰余演算器では、モンゴメリ演算の補正演算中に初期値を設定することにより単位元を最初に設定する必要がなくなる。このため、入力データの依存の有無が現れる電流消費のパターンとなって最初に0が連続する個数が見破られることを困難にできる。

#### 【0220】

請求項5記載のべき乗剰余演算器では、電流変動パターンから累積加算レジスタに格納されるべき値には、最下位から連続する0が何個あるかが見破られることを困難にすることが可能である。

#### 【0221】

請求項6～8記載のべき乗剰余演算器では、補正が不要の場合も、結果をレジスタに書き込むという動作が必ず行われるので、レジスタへの書き込みの有無より消費電流に差が生じることを回避できる。このため、電力解析により外部から処理を推定することが困難となる。

#### 【図面の簡単な説明】

【図1】 べき乗剰余演算器の演算を高速化するための第1の構成を説明するための概略ブロック図である。

【図2】 べき乗剰余演算器の演算を高速化するための第2の構成を説明するための概略ブロック図である。

【図3】 実施の形態1の乗剰余演算器1200の構成を説明するための概略ブロック図である。

【図4】 本発明の実施の形態2の乗剰余演算器1300の構成を説明するための概略ブロック図である。

【図5】 本発明の実施の形態3の乗剰余演算器1400の構成を説明するための概略ブロック図である。

【図6】 本発明の実施の形態4の乗剰余演算器1500の構成を説明するための概略ブロック図である。

【図7】 本発明の実施の形態5の乗剰余演算器1600の構成を説明するための概略ブロック図である。

【図8】 本発明の実施の形態6の乗剰余演算器1700の構成を説明するための概略ブロック図である。

【図9】 本発明の実施の形態7の乗剰余演算器1800の構成を説明するための概略ブロック図である。

【図10】 本発明の実施の形態8の乗剰余演算器1900の構成を説明するための概略ブロック図である。

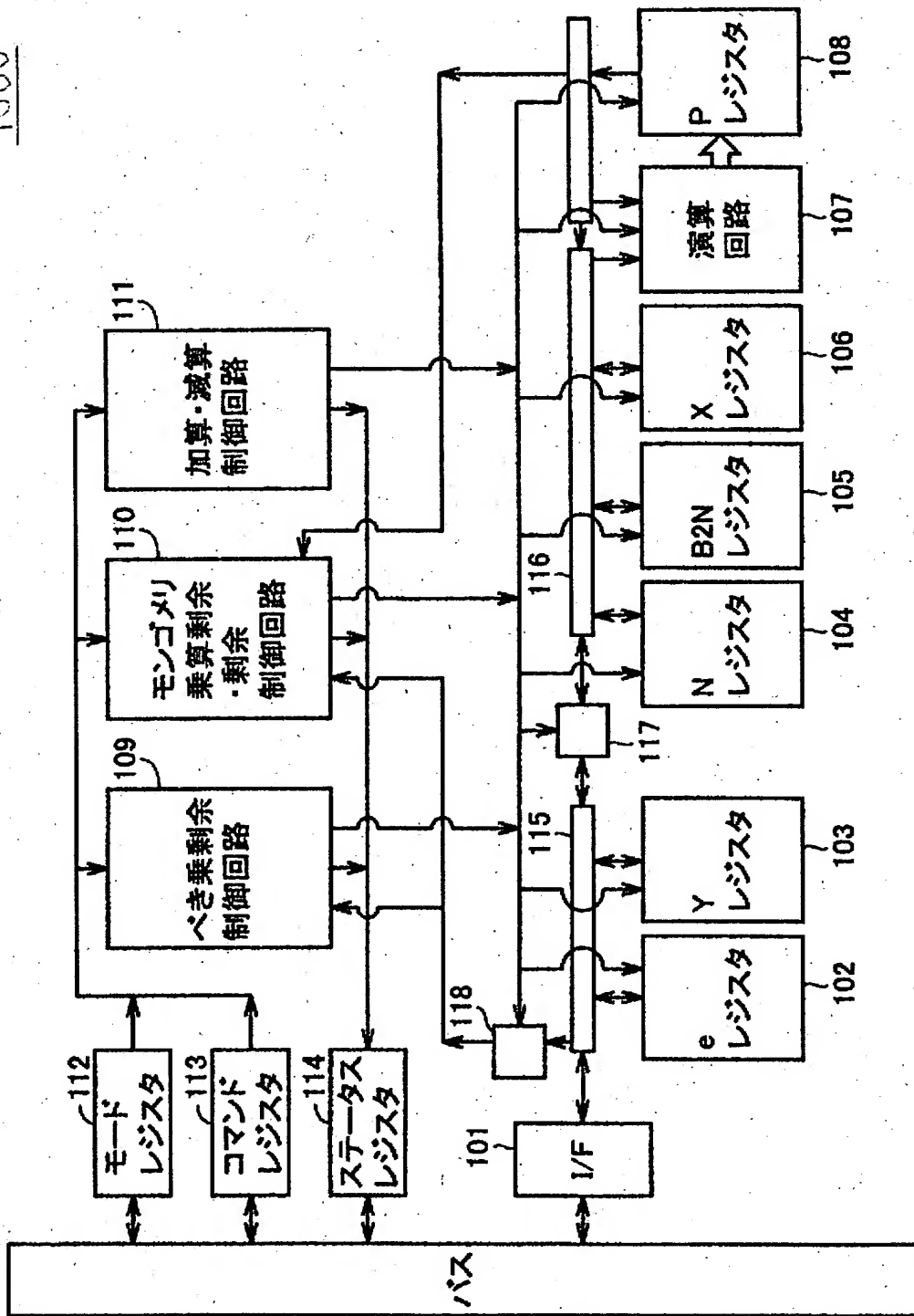
【符号の説明】

101 インタフェース回路、102 eレジスタ、103 Yレジスタ、104 Nレジスタ、105 B2Nレジスタ、106 Xレジスタ、107 演算回路、108 Pレジスタ、109 べき乗剰余制御回路、110 モンゴメリ乗算剰余・剰余制御回路、111 加算・減算制御回路、112 モードレジスタ、113 コマンドレジスタ、114 ステータスレジスタ、115, 116 内部バス、117 バス分割回路、118  $E_j/Y_j$  検出部、201, 202, 203, 206 Dフリップフロップ、204, 207 NANDゲート、204, 205, 208 インバータ、209 NORゲート。

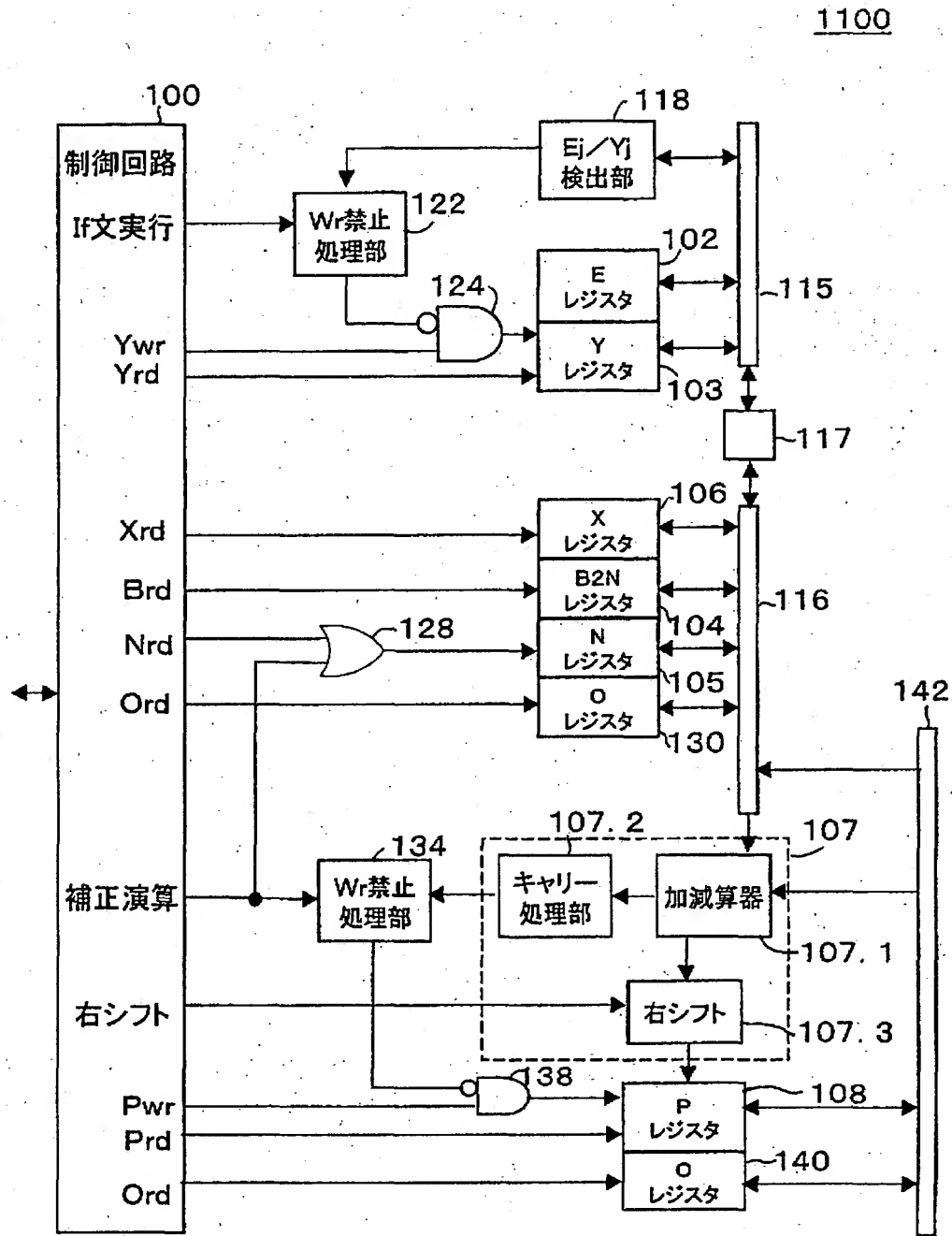
【書類名】 図面

【図 1】

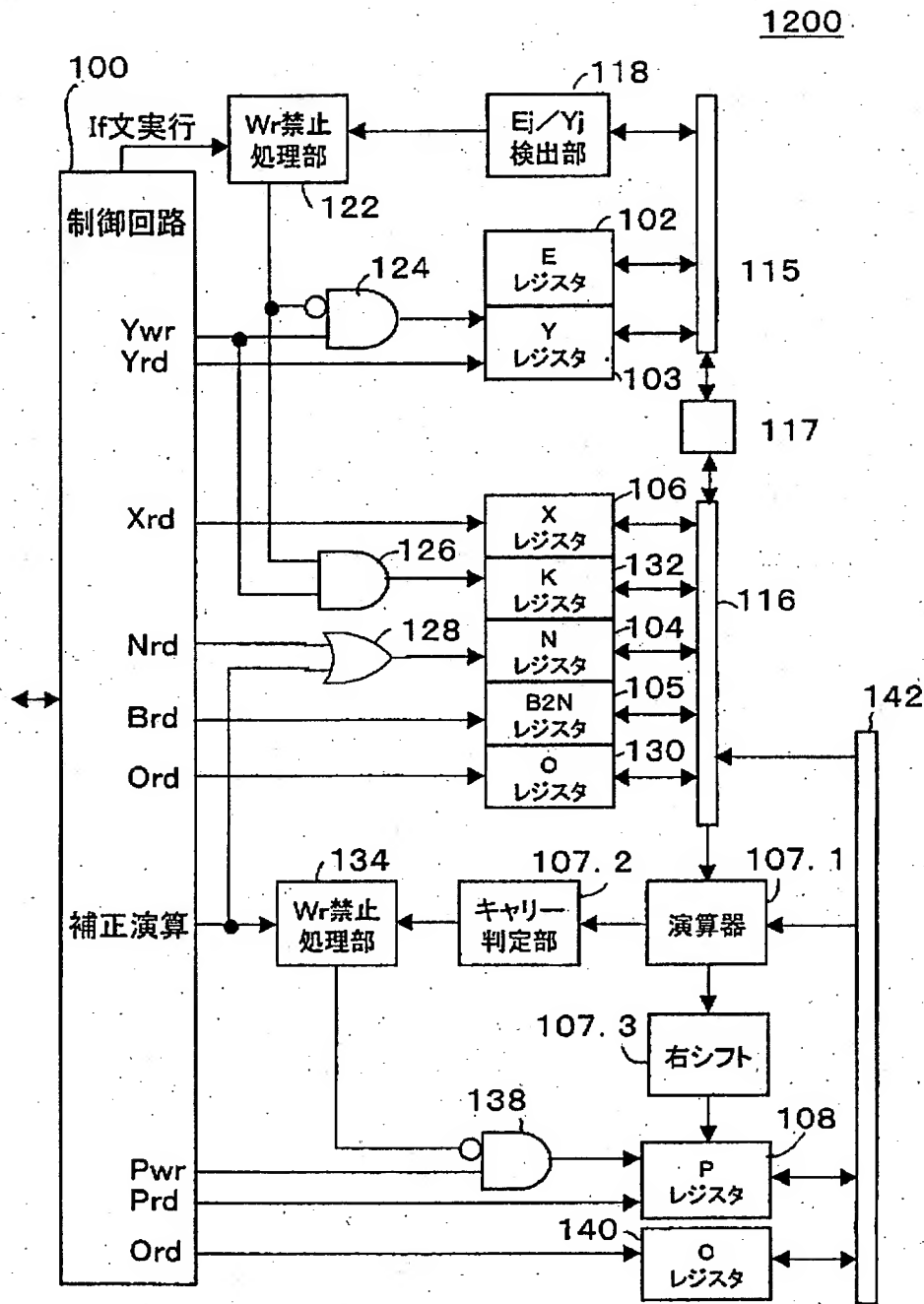
1000



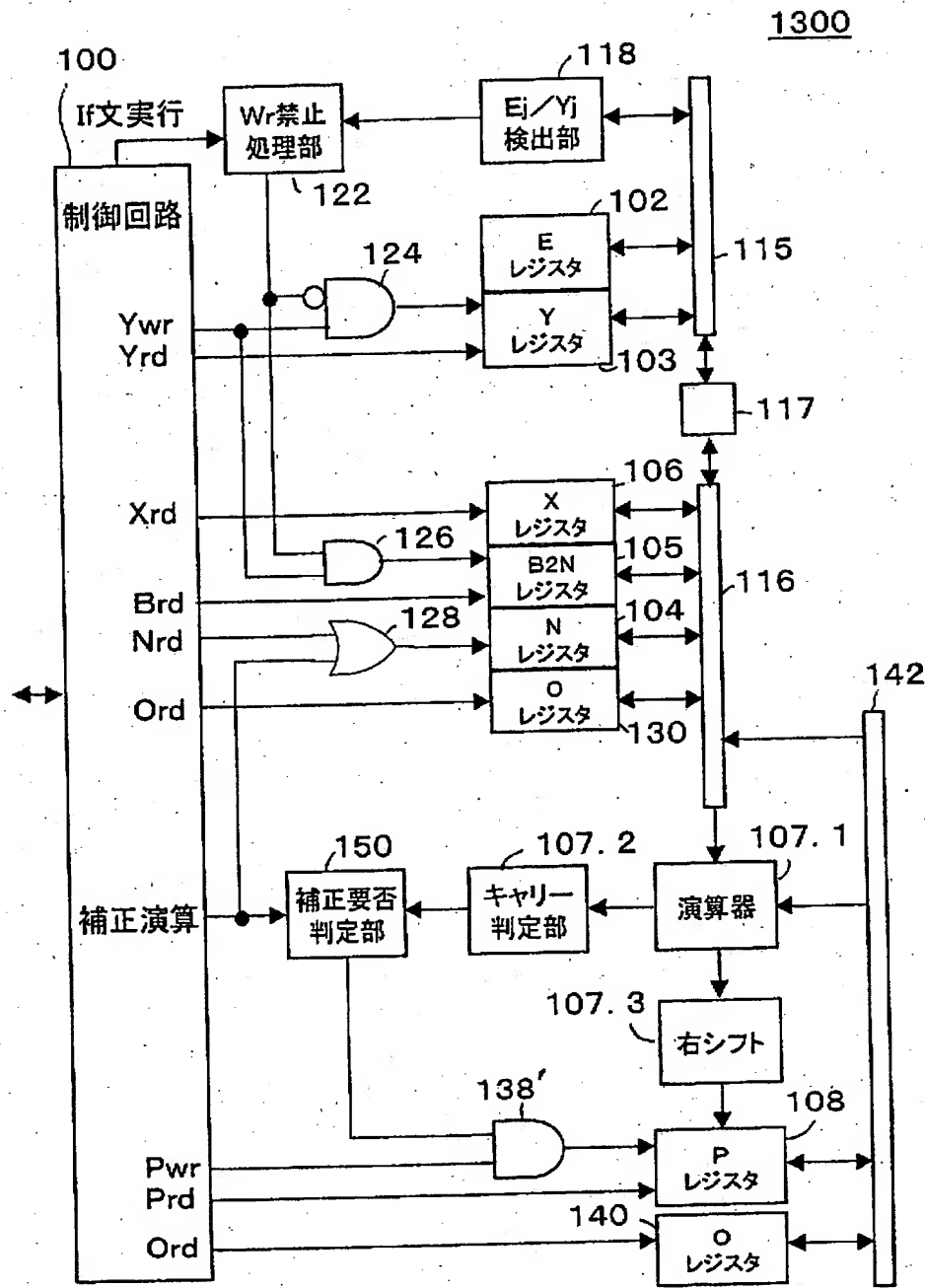
【図 2】



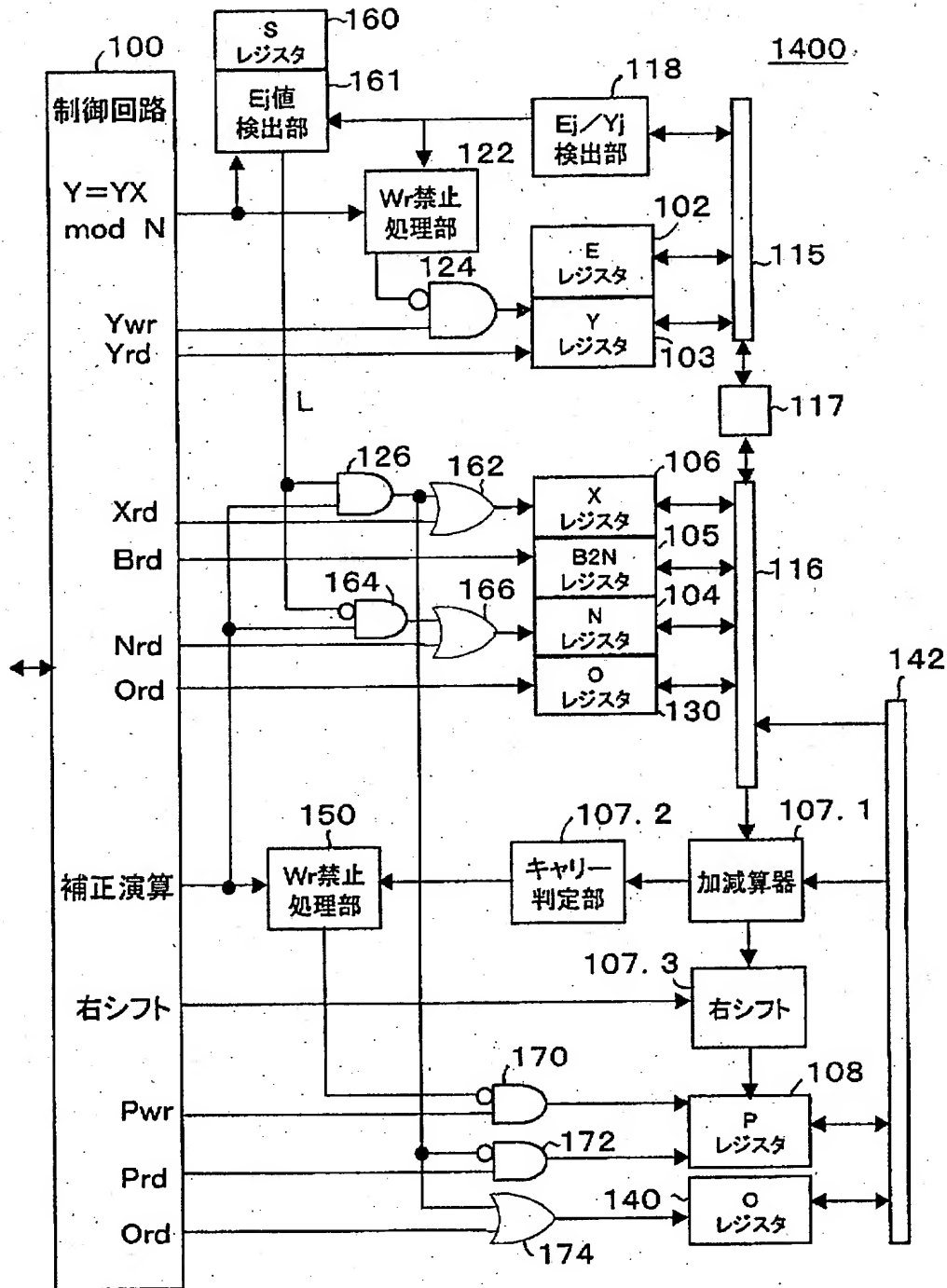
【図 3】



【図4】

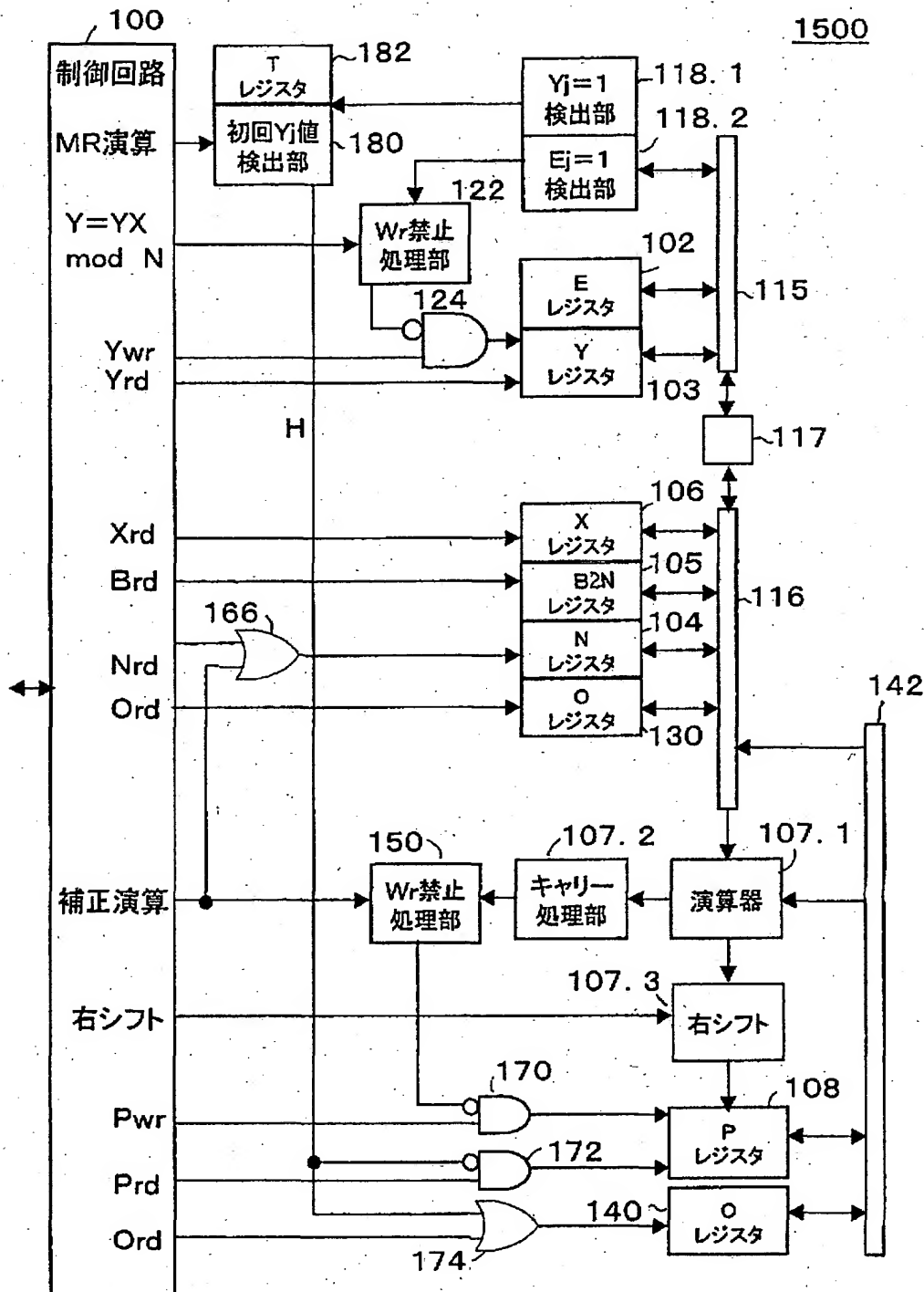


【図 5】

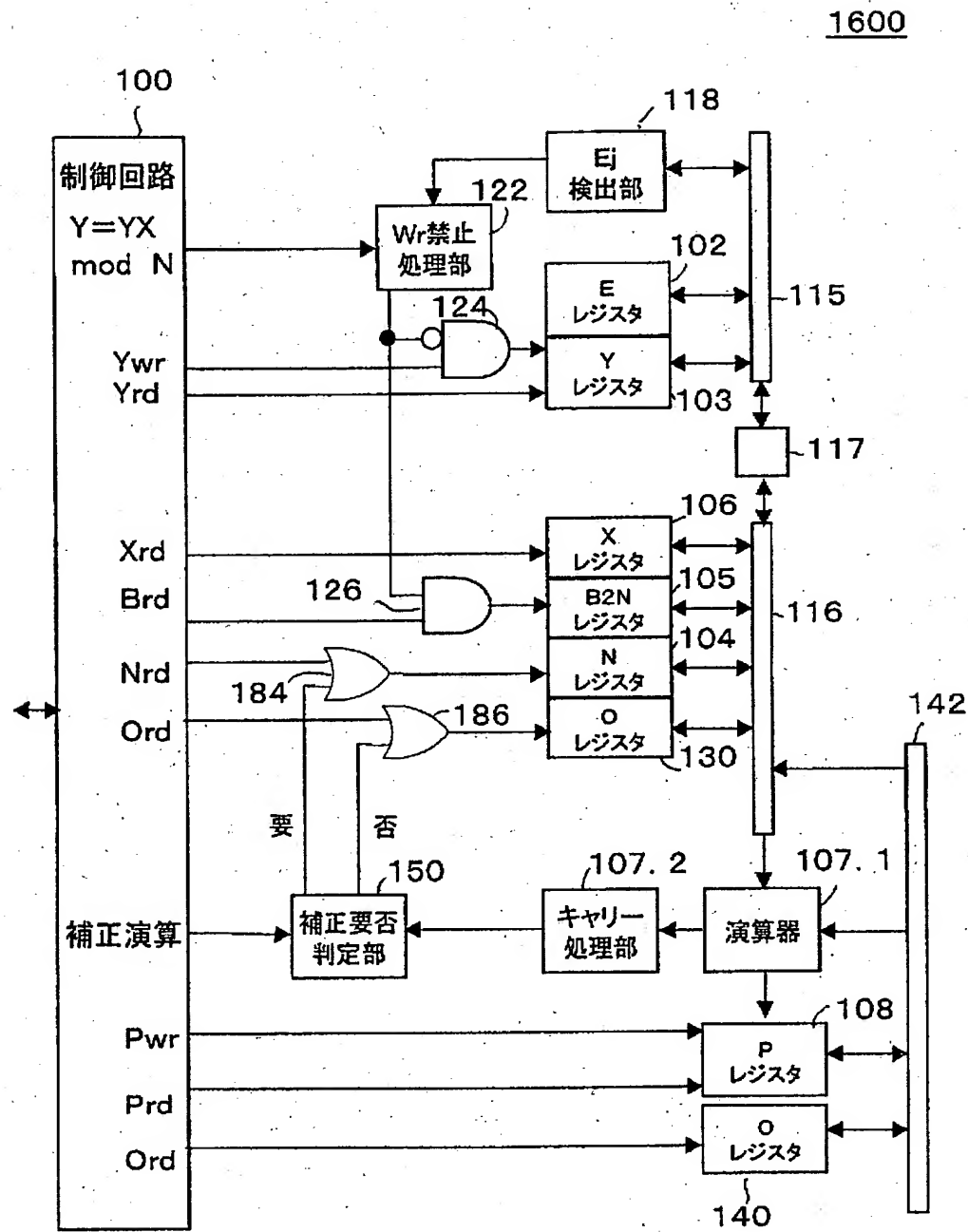




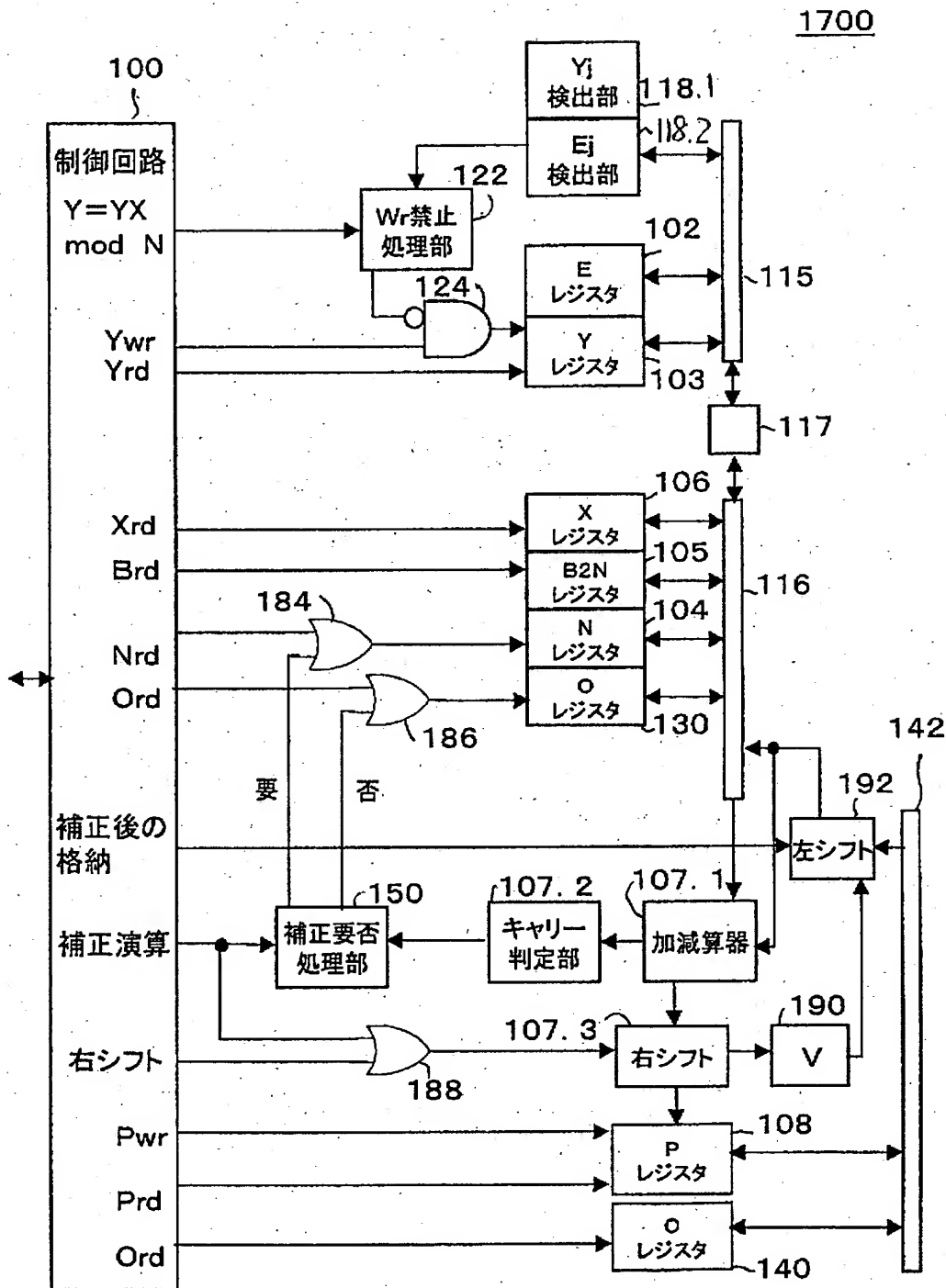
【図 6】



【図 7】

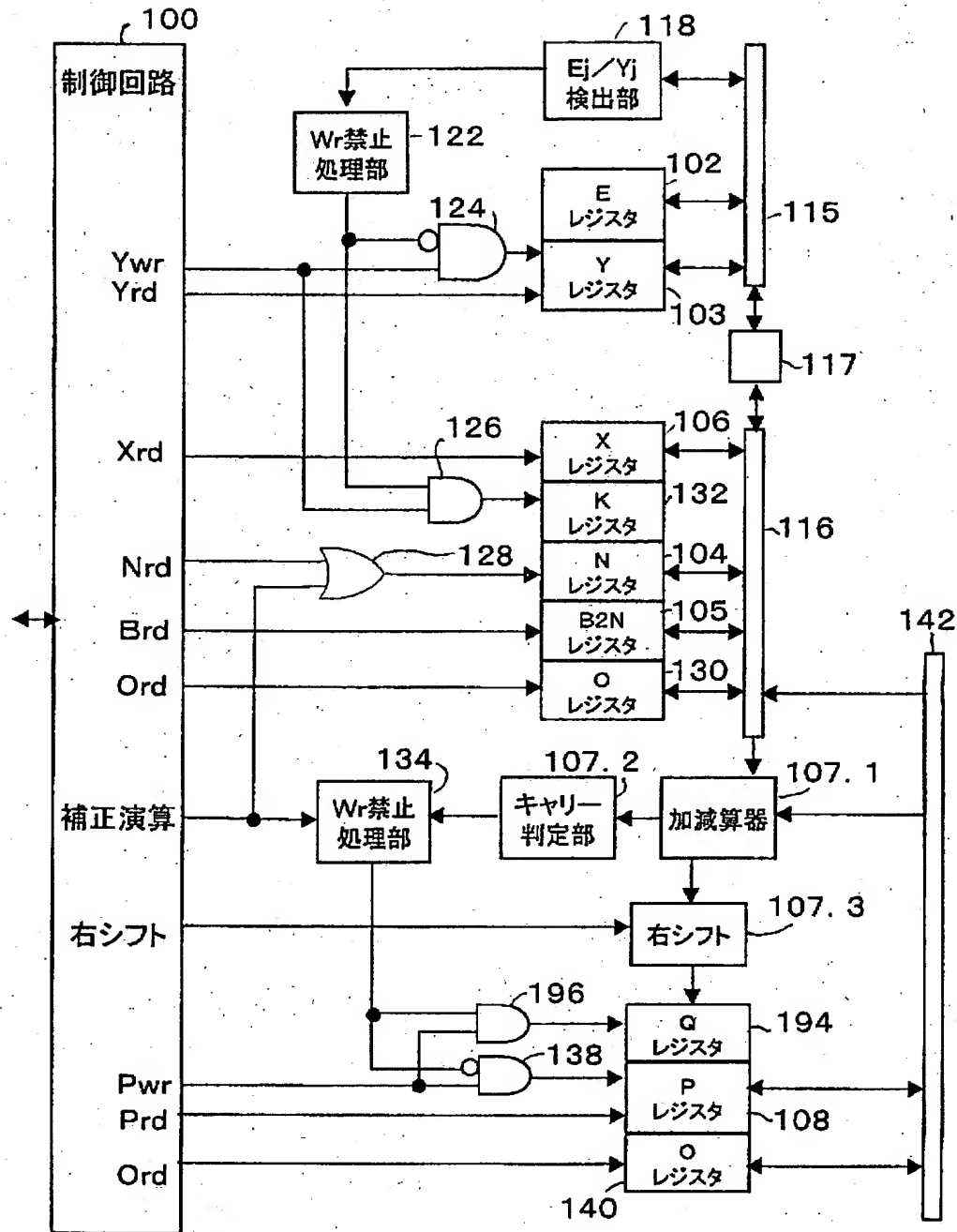


【図8】

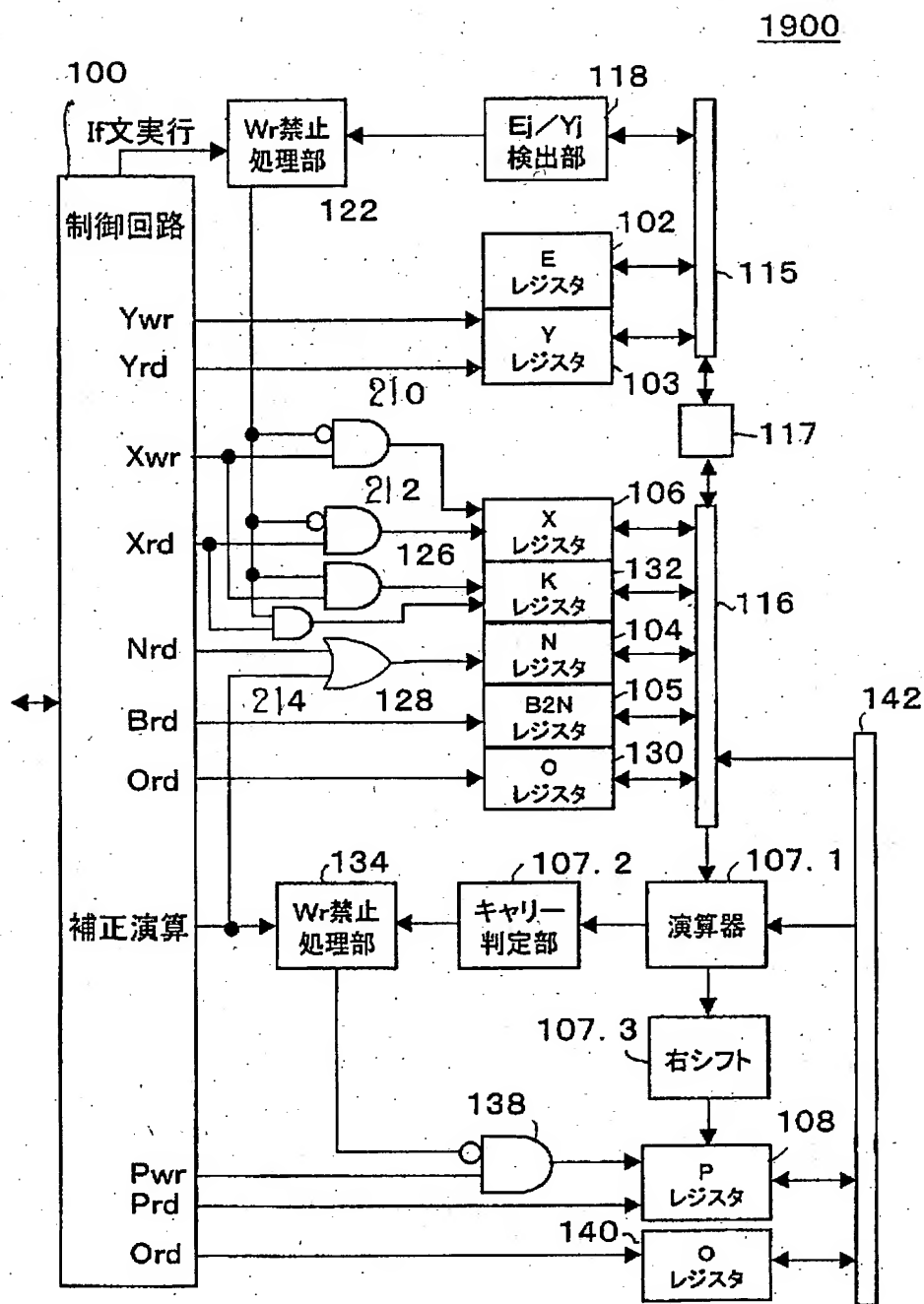


【図 9】

1800



【圖 10】



【書類名】 要約書

【要約】

【課題】 回路規模が小さく、高速処理が可能であって、かつ、電力解析に対する安全性を高めた暗号処理回路等に適用可能なべき乗剰余演算器を提供する。

【解決手段】 乗剰余演算器 1200 は、第 1 の内部バス 116 に接続され、バイナリ法に従うべき乗剰余演算を実行する際に、破棄されるべき中間演算結果を一旦格納するための K レジスタ 132 を備える。このため、破棄されるデータが計算途中に現れた場合でも K レジスタ 132 への書き込みが行われるので、書き込み時の電流が流れることとなり、電力解析に対する耐性が向上する。

【選択図】 図 5

出 願 人 履 歴 情 報

識別番号

[000006013]

1. 変更年月日 1990年 8月24日

[変更理由] 新規登録

住 所 東京都千代田区丸の内2丁目2番3号

氏 名 三菱電機株式会社